

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

## - Technical Report -

Jeongho Kwak<sup>†</sup>, Okyoung Choi<sup>†</sup>, Song Chong<sup>†</sup> and Prasant Mohapatra<sup>‡</sup>

<sup>†</sup>Department of Electrical Engineering, KAIST

<sup>‡</sup>Department of Computer Science, University of California, Davis

E-mail: {jh.kwak, okyoung}@netsys.kaist.ac.kr, songchong@kaist.edu, pmohapatra@ucdavis.edu

### Abstract

Many smartphone applications such as video upload and file backup are intrinsically delay-tolerant so that it is possible to delay data processing and transfer to a certain extent in order to save smartphone battery. In the literature these energy-delay tradeoff issues have been addressed independently in the forms of DVFS (Dynamic Voltage and Frequency Scaling) problem and network selection problem when smartphones have multiple wireless interfaces, 3G and WiFi. In this paper, we jointly optimize CPU speed control and network(wireless interface) selection so as to answer how much energy can be saved further by the joint optimization when applications can tolerate a certain delay. In addition, since contemporary smartphones with multitasking capability are likely to simultaneously run networking and non-networking applications, the heterogeneity of applications should be also considered in the joint optimization so as to isolate the performance of nonnetworking applications from that of networking applications. We propose a dynamic speed scaling scheme called *SpeedControl* which jointly adjusts both processing and networking speeds by three control knobs: application scheduling, CPU speed control and wireless interface selection. By invoking the Lyapunov drift plus penalty method, the scheme is shown to be near-optimal in the sense that it minimizes energy consumption for a given delay constraint. This paper is the first to reveal energy-delay tradeoff relationship in a holistic view for smartphones with multiple wireless interfaces, DVFS and multitasking capability. We perform real measurements on WiFi/3G temporal coverage and data rate in a metropolitan area, power consumption of CPU and WiFi/3G interfaces in popular smartphone models, and file size distribution and CPU workload of popular videos. Tracedriven simulations based on the measurements demonstrate that SpeedControl 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 increases as WiFi temporal coverage increases.

## I. INTRODUCTION

Energy consumption in processing and transferring data in smartphone applications 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 one bit for typical encoding applications is comparable with that of WiFi or 3G interface to transmit one 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 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. Earlier works studied energy minimization in cellular networks, processor sharing systems and routing under DVFS [6]–[8].

For energy minimization in a smartphone with multiple network interfaces, previous works [9], [10] proposed energy-efficient network selection policies under heterogeneous wireless network interfaces (e.g., 3G, LTE 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 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 [11], 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 [12], [13]). 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 [14], 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 of Seoul, power consumption of CPU and WiFi/3G interfaces in popular smartphone models (Nexus S and Galaxy Nexus), and file size distribution and CPU workloads (measured by processing density in unit of cycles/bit) for popular video clips including YouTube video files.

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 a given delay constraint.
- 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 increases as WiFi temporal coverage increases.

In the rest of this paper, we begin with related work in Section II. In Section III, 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 [6]–[8]. Son *et. al.* [6] 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. Wierman *et. al.* [7] analyze the optimal energy-delay tradeoff under several processing power models including static and dynamic speed scaling. They show that dynamic speed scaling significantly improves robustness to bursty traffic.

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 exist 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 optimal and inefficient, e.g., in Ondemand policy [15], 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 [9], [10], [16], [17], have been increasing. Rahmati *et. al.* [16] 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 [9], [10] suggest delayed network selection by exploring tradeoff between transmit power of heterogeneous network interfaces (3G, WiFi) and transmission delay. Lee *et. al.* [17] show 20% power saving can be achieved by permitting 1 hour delayed WiFi offloading for application having strict delay constraint.

In the rest of this paper, we begin with measurement study of CPU and network module in Section ???. In Section II, we propose a joint application scheduling, CPU speed and Network selection (SpeedControl) algorithm. Then, in Section III, we show the design of our SpeedControl algorithm. Next, in Section IV-B, we extensively evaluate our SpeedControl algorithm under the real traces. Finally, we conclude this paper in Section V.

## 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 selection, called SpeedControl, algorithm.

### A. Model and Problem Formulation

**Arrival Model.** We consider networking applications which use uplink wireless networks and non-networking applications which do not use the network in a smartphone, and two types of applications are delay tolerant applications. For example, 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  $\mathcal{K} = \{1, \dots, K\}$ . For simplicity, we consider one NA and another NNA. However, it can be easily generalized to multiple applications. We consider a time-slotted system indexed by  $t = \{0, 1, \dots\}$  where the interval is  $\Delta t$ . For each time slot  $t$ , workload  $A_{NA}(t)$  for NA, and  $A_{NNA}(t)$  for NNA are arrived with average rate  $\lambda_{NA}$  and  $\lambda_{NNA}$ , respectively.

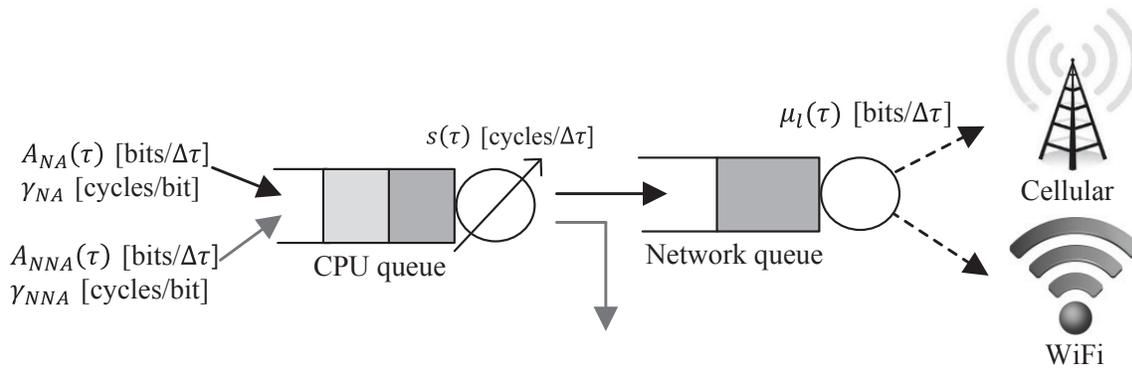


Fig. 1: Queueing model for SC-NoAS system

**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)  $\gamma_{NA}$  for NA,  $\gamma_{NNA}$  for NNA which are defined as the average number of CPU cycles required per bit when the application is processed by CPU [18]. We assume that the smartphone can adjust CPU speed  $s(t) \in \{s_1, s_2, \dots, s_{max}\}$  (in cycles/ $\Delta t$ ) 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  $l(t)$  varies along with dynamics of wireless channel states. Also, we assume that the smartphone can select no network transmission, or cellular or WiFi networks,  $l(t) \in \{N, C, W\}$  every time slot  $t$ .

### Queueing Model.

For given arrival, CPU and network models, we consider a queueing model as illustrated in Fig. 1. In this queue model, CPU schedules the application by first in first out (FIFO) manner. For the queueing model,  $Q^c(t)$  denotes total CPU queue length at time slot  $t$ ,  $Q_{NA}^c(t)$  and  $Q_{NNA}^c(t)$  denote CPU queue lengths for NA and NNA at time slot  $t$ , respectively. Thus,  $Q^c(t) = Q_{NA}^c(t) + Q_{NNA}^c(t)$ . Also,  $Q^n(t)$  denotes network queue length at time slot  $t$ . We assume that the unit of queue lengths is a bit, thus the CPU speed  $s(t)$  (in cycles/ $\Delta t$ ) should be divided by processing density (cycles/bit)  $\gamma_{NA}$  or  $\gamma_{NNA}$  in the queue model. Then, queue lengths of all CPU and network queues are updated as follows.

$$Q^c(t+1) = [Q^c(t) - \phi(t) \frac{s(t)}{\gamma_{NA}} + A_{NA}(t) + A_{NNA}(t)]^+ [\text{bits}] \quad (1)$$

$$Q^n(t+1) = [Q^n(t) - \mu_l(t) + \min\{Q^c(t), \phi(t)s(t)\}]^+ [\text{bits}] \quad (2)$$

$$\phi(t) = \frac{Q_{NA}^c(t)\gamma_{NA}}{Q_{NA}^c(t)\gamma_{NA} + Q_{NNA}^c(t)\gamma_{NNA}}, \quad (3)$$

**Power Model.** CPU power consumption can be modeled in general as a follow [8].

$$P^c(s(t)) = \alpha s(t)^3 + \beta \quad (4)$$

where  $\alpha$  and  $\beta$  are constants depending on different smartphones. We assume that the network power depends on the selected network, so the network power model is defined as follows [9].

$$P^n(l(t)) \in \{P^n(N), P^n(C), P^n(W)\} \quad (5)$$

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

**Problem Formulation.** Our objective under the queueing model in Fig. 1 is to develop an energy-minimal joint CPU speed adjustment and network selection policy, called SC-NoAS. 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

$$(\mathbf{P}) : \quad \min \bar{P} = \lim_{T \rightarrow \infty} \frac{1}{T} \sum_{t=0}^{T-1} (P^c(t) + P^n(t)), \quad (6)$$

$$s.t. \quad \limsup_{t \rightarrow \infty} \frac{1}{t} \sum_{\tau=0}^{t-1} \mathbb{E}\{Q^c(\tau) + Q^n(\tau)\} < \infty, \quad (7)$$

The constraint means that CPU and network queue lengths should be finitely maintained [14]. For the above problem **(P)**, we determine the CPU speed  $s(t)$  and network selection  $l(t)$  every time slot  $t$ .

We obtain the solution of our problem **(P)** under unknown future sequences of wireless network states and data arrival in inheriting Lyapunov drift plus penalty technique [14].

**SC-NoAS algorithm** The SC-NoAS algorithm can be described as a follow.

---

**SC-NoAS Algorithm**

---

1: For every time slot  $t$ ,

2: **if**  $Q^n(t) \geq \mu_{max}(t)$ , **then**

4: Select CPU speed  $s^*(t)$  by

$$\min_{s^*(t)} \left\{ VP^c(s(t)) - (1 - \phi(t)) \frac{s(t)}{\gamma_{NNA}} (Q^c(t) + Q^n(t)) \right\} \quad (8)$$

5: Select network interface  $l^*(t)$  by

$$\min_{l^*(t)} \left\{ VP^n(l(t)) - \mu_l(t) (Q^c(t) + Q^n(t)) \right\} \quad (9)$$

6: **else**

7: Select CPU speed  $s^*(t)$  by

$$\min_{s^*(t)} \left\{ VP^c(s(t)) - (Q^c(t) + Q^n(t)) \left( \frac{\phi(t)}{\gamma_{NA}} + \frac{1 - \phi(t)}{\gamma_{NNA}} \right) s(t) \right\} \quad (10)$$

8: Select network interface  $l^*(t)$  by

$$\min_{l^*(t)} \left\{ VP^n(l(t)) - \mu_l(t) (Q^c(t) + Q^n(t)), \right. \quad (11)$$

$$\left. \mu_l(t) \leq Q^n(t) + (1 - \phi(t))s(t), \right. \quad (12)$$

9: **end all**

---

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

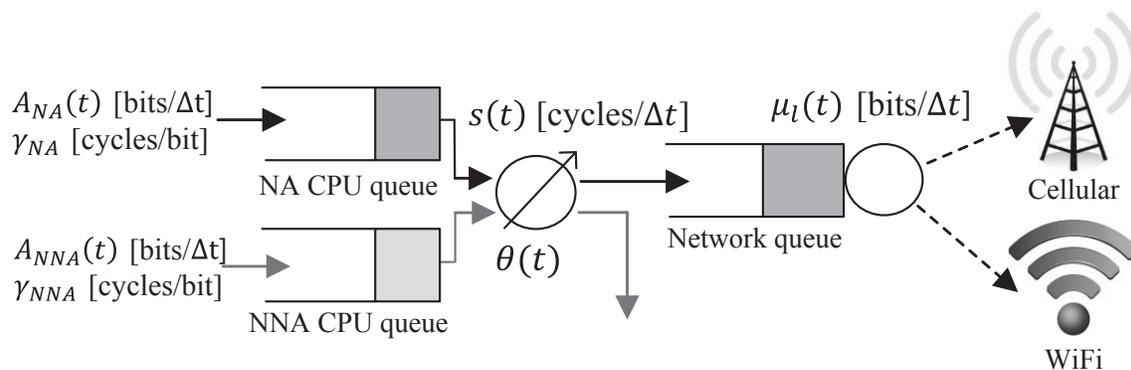


Fig. 2: Queuing model for SpeedControl system

However, SC-NoAS for the queueing model has a interference problem between NA and NNA. NNA should not be affected by the network environment since it does not use network part. However, in the queueing model of Fig. 1, 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 resources. This is the reason why NA and NNA workloads should be distinguished and processed separately. Therefore, henceforth, we consider dual CPU queue model where NA and NNA workload are distinguished and processed separately in Fig. 2. In this model, we consider application scheduling  $\theta(t) \in \{0, 1\}$ , which denotes scheduling indicator by CPU processor at time slot  $t$ , e.g., if  $\theta(t)$  is 0, NNA is scheduled. Also, queueing dynamics are also redesigned as follows.

$$Q_{NA}^c(t+1) = \left[ Q_{NA}^c(t) - \frac{\theta(t)s(t)}{\gamma_{NA}} + A_{NA}(t) \right]^+ \text{ [bits]} \quad (13)$$

$$Q_{NNA}^c(t+1) = \left[ Q_{NNA}^c(t) - \frac{(1-\theta(t))s(t)}{\gamma_{NNA}} + A_{NNA}(t) \right]^+ \text{ [bits]} \quad (14)$$

$$Q^n(t+1) = \left[ Q^n(t) - \mu_l(t) + \theta(t) \min \left\{ Q_{NA}^c(t), \frac{s(t)}{\gamma_{NA}} \right\} \right]^+ \text{ [bits]} \quad (15)$$

Then,  $Q^c(t) = Q_{NA}^c(t) + Q_{NNA}^c(t)$ .

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

We obtain the solution of our problem **(P)** for redesigned queueing model under unknown future sequences of wireless network states and data arrival in inheriting Lyapunov drift plus penalty technique [14]. We call the redesigned algorithm as *SpeedControl* algorithm, which is to jointly control application scheduling, CPU speed adjustment and network selection satisfying our problem **(P)**.

**Algorithm Description.** The SpeedControl algorithm can be described as a follow.

---

**SpeedControl Algorithm**

---

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

$$\min_{s^*(t)} \left\{ VP^c(s(t)) - \frac{s(t)}{\gamma_{NNA}} Q_{NNA}^c(t) \right\} \quad (16)$$

- 5:   Select network  $l^*(t)$  by

$$\min_{l^*(t)} \left\{ VP^n(l(t)) - \mu_l(t)(Q_{NA}^c(t) + Q^n(t)) \right\} \quad (17)$$

- 6: **else**

- 7:   **if**  $\frac{Q_{NA}^c(t) + Q^n(t)}{\gamma_{NA}} \geq \frac{Q_{NNA}^c(t)}{\gamma_{NNA}}$ , **then**

- 8:     Schedule NA ( $\theta^*(t) = 1$ )
- 9:     Select CPU speed  $s^*(t)$  by

$$\min_{s^*(t)} \left\{ VP^c(s(t)) - \frac{s(t)}{\gamma_{NA}} (Q_{NA}^c(t) + Q^n(t)) \right\} \quad (18)$$

- 10:    Select network  $l^*(t)$  by (17)

- 11: **else**

- 12:    Schedule NNA ( $\theta^*(t) = 0$ )
- 13:    Select CPU speed  $s^*(t)$  by (16)
- 14:    Do not select network ( $l(t) = N$ )

- 15: **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 (16)-(18) can be interpreted as follows. An energy-delay tradeoff can be controlled by single parameter  $V$ , i.e., as  $V$  is larger, energy consumption is lower by trading longer delay. Also, the first term with  $V$  of each problem (16)-(18) minimizes CPU or network power consumption, and remained terms without  $V$  minimize average delay, i.e., strive to stabilize CPU and/or network queue. If  $V$  is small and queues are increased 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 are higher due to the high variation of  $s(t)$  and the fact that the system is likely to immediately transmit without consideration of energy-efficiency of network interface (e.g., 3G with low throughput). Therefore, the system makes more effort to reduce delay than to reduce energy. On the other hand, if  $V$  is large and queues are increased 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 uses  $s(t)$  more smoothly and is likely to defer transmission or select energy-efficient networks (e.g., WiFi network). Therefore, the problems (16)-(18) make more effort to reduce CPU and network energy than to reduce delay.

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 delay 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 delay 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 our SpeedControl algorithms in inheriting Lyapunov drift plus penalty function [14].

**Making Single Objective.** Our original objective is to minimize CPU and network power consumption with queueing stability in (6), (7). We first define Lyapunov function and Lyapunov drift function as follows.

$$L(t) \triangleq \frac{1}{2}[Q_{NA}^c(t) + Q^n(t)]^2 + \frac{1}{2}[Q_{NNA}^c(t)]^2 \quad (19)$$

$$\Delta(L(t)) \triangleq \mathbb{E}\{L(t+1) - L(t) | \mathbf{Q}(t)\} \quad (20)$$

$$\mathbf{Q}(t) = \{Q_{NNA}^c(t), Q_{NA}^c(t), Q^n(t)\} \quad (21)$$

The Lyapunov function (19) is designed to fairly stabilize NA queues ( $Q_{NA}^c(t) + Q^n(t)$ ) and NNA queue ( $Q_{NNA}^c(t)$ ). This is a key design principle to differentiate NA and NNA in terms of scheduling. Under this Lyapunov function design, if the network queue is accumulated due to network bottleneck, scheduling NNA would not reduce NA queues ( $Q_{NA}^c(t) + Q^n(t)$ ), so NNA would be always scheduled to reduce Lyapunov drift function (20). On the other hand, if the network queue is not accumulated, NA and NNA would be fairly scheduled in terms of queue lengths<sup>1</sup>. This is just a criterion to differentiate NA and NNA

<sup>1</sup>In fact, since the units of processing speed (in cycles/ $\Delta t$ ) and traffic in the queue (in bits) are different, processing density (in cycles/bit) of application should be also considered.

discussed in Section II-A.

Next, we define drift plus penalty function which considers expected CPU and network power consumption during time slot  $t$  as  $\mathbb{E}\{P^c(s(t))|\mathbf{Q}(t)\}$  and  $\mathbb{E}\{P^n(l(t))|\mathbf{Q}(t)\}$  in inheriting drift-plus-penalty expression [14] as follows.

$$\Delta(L(t)) + V\mathbb{E}\left\{P^c(s(t)) + P^n(l(t))|\mathbf{Q}(t)\right\} \quad (22)$$

where  $V$  is energy-delay tradeoff parameter. Then, our single objective is to minimize the equation (22).

**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.

$$A_{NA}(t) \leq A_{NA,max}, \quad A_{NNA}(t) \leq A_{NNA,max} \quad (23)$$

$$s(t) \leq s_{max}, \quad \mu_l(t) \leq \mu_{max} \quad (24)$$

These bounded values and queueing dynamic equations in (13)-(15) make drift plus penalty function (22) be bounded as a following Lemma.

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

$$\begin{aligned} & \Delta(L(t)) + V\mathbb{E}\left\{P^c(s(t)) + P^n(l(t))|\mathbf{Q}(t)\right\} \leq \\ & B + V\mathbb{E}\left\{P^c(s(t)) + P^n(l(t))|\mathbf{Q}(t)\right\} - \mathbb{E}\left\{Q_{NNA}^c(t) \left(\frac{(1-\theta(t))s(t)}{\gamma_{NNA}} - A_{NNA}(t)\right) |\mathbf{Q}(t)\right\} \\ & - \mathbb{E}\left\{(Q_{NA}^c(t) + Q^n(t)) \times \left(\min\left\{\frac{\theta(t)s(t)}{\gamma_{NA}} + Q^n(t), \mu_l(t)\right\} - A_{NA}(t)\right) |\mathbf{Q}(t)\right\}, \end{aligned} \quad (25)$$

where  $B = \frac{1}{2}\left(\frac{s_{max}^2}{\gamma_{NNA}} + \mu_{max}^2 + A_{NA,max}^2 + A_{NNA,max}^2\right)$ .

*Proof:* Please refer to Appendix. ■

**Deriving Solution.** Minimizing the left-hand side of (25) means that our original problem **(P)** is satisfied. From Lemma 1, we can derive SpeedControl algorithm. First, we show that the problem **(P)** has an optimal policy  $\pi^*$  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  $\Lambda$  denotes all mean arrival workloads that the smartphone can process within finite time, there exists a stationary randomized control policy  $\pi^*$  that selects application scheduling*

$\theta(t)$ , CPU speed  $s(t)$ , and network  $l(t)$  every time slot  $t$  satisfying the following:

$$\mathbb{E}\{P^c(s(t))^{\pi^*}\} = P^c(\lambda_{NA} + \lambda_{NNA}) \quad (26)$$

$$\mathbb{E}\{P^n(l(t))^{\pi^*}\} = P^n(\lambda_{NA}) \quad (27)$$

$$\mathbb{E}\left\{\frac{\theta(t)^{\pi^*} s(t)^{\pi^*}}{\gamma_{NA}}\right\} = \mathbb{E}\{\mu_l(t)\} \quad (28)$$

$$\mathbb{E}\{A_{NA}(t)\} = \mathbb{E}\left\{\frac{\theta(t)^{\pi^*} s(t)^{\pi^*}}{\gamma_{NA}}\right\} \quad (29)$$

$$\mathbb{E}\{A_{NNA}(t)\} = \mathbb{E}\left\{\frac{(1 - \theta(t)^{\pi^*}) s(t)^{\pi^*}}{\gamma_{NNA}}\right\} \quad (30)$$

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

*Proof:* It can be proven using Caratheodory's theorem in [19]. ■

Then, SpeedControl algorithm should be made to minimize the left-hand side of (25) compared to the other alternative policies, i.e., the SpeedControl algorithm makes the right-hand side of (25) the smallest value among the values which obtains from all possible stationary randomized control policies.

$$\begin{aligned} & \Delta L(t) + V\mathbb{E}\{P^{c,SC}(t) + P^{n,SC}(t) | \mathbf{Q}(t)\} \\ & \leq B' + V\mathbb{E}\{P^{c,alt}(t) + p^{n,alt}(t) | \mathbf{Q}(t)\} - \mathbb{E}\left\{Q_{NNA}^c(t) \frac{(1 - \theta(t)^{alt}) s(t)^{alt}}{\gamma_{NNA}} | \mathbf{Q}(t)\right\} \\ & - \mathbb{E}\left\{(Q_{NA}^c(t) + Q^n(t)) \times \min\left\{\frac{\theta(t)^{alt} s(t)^{alt}}{\gamma_{NA}} + Q^n(t), \mu_l^{alt}(t)\right\} | \mathbf{Q}(t)\right\}, \end{aligned} \quad (31)$$

where *alt* denotes any alternative control policy which determines application scheduling  $\theta(t)$ , CPU speed  $s(t)$  and network selection  $l(t)$ . Also,  $B' = B + \mathbb{E}\{A_{NNA}(t)Q_{NNA}^c(t) + (Q_{NA}^c + Q^n)A_{NA} | \mathbf{Q}(t)\}$ .

Then, an optimal algorithm which minimizes (22) is to find control variables  $(\theta(t), s(t), l(t))$  which minimize the left-hand side of (31), i.e., the optimal algorithm makes the right-hand side of (31) the smallest value among the values which obtains from all possible stationary randomized control policies. Then, the right-hand side of (31) can be decomposed into application scheduling  $\theta(t)$ , CPU speed  $s(t)$  and network selection  $l(t)$  problems 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 henceforth sequentially approximate the optimal algorithm. First of all, we take following reasonable approximation.

$$Q^n(t) \approx 0, \text{ for } Q^n(t) < \mu_{max}(t) \quad (32)$$

Then, if  $Q_{NNA}^c(t) \frac{s(t)}{\gamma_{NNA}} > Q_{NA}^c(t) \frac{s(t)}{\gamma_{NA}}$ , the right-hand side of (25) can be decomposed into  $(\theta(t) = 0, s(t), l(t) = N)$  such as (16). In the other case, however, we should address complicated problems as follows.

$$\min_{s(t)} V(P^c(s(t)) + P^n(N)) - \frac{s(t)}{\gamma_{NNA}} Q_{NNA}^c(t) \quad (33)$$

$$\min_{s(t), l(t)} V(P^c(s(t)) + P^n(l(t))) - \min\left(\frac{s(t)}{\gamma_{NA}}, \mu_l(t)\right) Q_{NA}^c(t) \quad (34)$$

If (33)  $\geq$  (34), then  $\theta(t) = 1$  and if (33)  $<$  (34), then  $\theta(t) = 0$ . Since jointly solving (33) and (34) is very 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 our SpeedControl algorithm. Then, SpeedControl algorithm selects CPU speed and network depending on the network bottleneck and application scheduling. Although our SpeedControl algorithm takes a few assumption, 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  $t = \{0, 1, \dots, T-1\}$ . 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, we have:*

$$\limsup_{T \rightarrow \infty} \frac{1}{T} \sum_{t=0}^{T-1} \mathbb{E}\{Q^c(t) + Q^n(t)\} \leq \frac{B + VP_{max}}{\epsilon}, \quad (35)$$

$$\bar{P}^{c,opt} + \bar{P}^{n,opt} \leq P^*(2\epsilon) + \frac{B}{V}. \quad (36)$$

where  $Q^c(t) = Q_{NA}^c(t) + Q_{NNA}^c(t)$ ,  $\epsilon = 2\epsilon' = \epsilon''$ ,  $P_{max} = P_{max}^c + P_{max}^n$  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 Appendix.* ■

The result of Theorem 2 can be interpreted as follows. As the energy-delay tradeoff parameter  $V$  is smaller, the sum of average queue lengths (NA and NNA) is smaller whereas the average power consumption is increased from the optimal lowest value  $P^*(2\epsilon)$ . On the other hand, as  $V$  is larger, the average CPU and network power consumptions are smaller whereas the sum of average queue lengths is larger. 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 our 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. The specification of each smartphone can be shown in Table I. We connect Monsoon power monitor [20] to experimental smartphones and measure the real power consumption of 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<sup>2</sup>. 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. 3 depicts the measured CPU power consumption over the CPU speed for five different smartphones. The diagrams 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. Interestingly, this is different polynomial with the power models of typical laptop processor or TCP offload engine [7] where the polynomial is closer to quadratic. From these results, we can set the CPU power consumption model over different CPU speed as a follow.

$$P^c(s(t)) = \alpha s(t)^3 + \beta \quad (37)$$

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

Also, we measure the power consumption of 3G and WiFi network interfaces using Monsoon power monitor [20] for two different android smartphones, respectively. For network interface measurement, we turned off running applications in a smartphone except for throughput measurement application which

<sup>2</sup>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.

|                       | Galaxy S        | Nexus S              | Galaxy Nexus            | Galaxy Note               | Galaxy S2               |
|-----------------------|-----------------|----------------------|-------------------------|---------------------------|-------------------------|
| Max. CPU speed        | 1GHz            | 1.44GHz(overclocked) | 1.2GHz                  | 1.51GHz(overclocked)      | 1.2GHz                  |
| Kernel                | Tegrak Build 24 | Trinity with 1.44GHz | CyanogenMod 9.0         | Tegrak Build 46           | Android Pure            |
| OS                    | Android v2.1    | Android v4.0         | Android v4.0            | Android v4.1.2            | Android v4.1.2          |
| # of CPU clock levels | 5               | 6                    | 4                       | 11                        | 5                       |
| AP                    | HummingBird     | Exynos3110           | Ti Omap4460 (dual core) | Snapdragon S3 (dual core) | Exynos 4210 (dual core) |

TABLE I: Specification of five smartphones

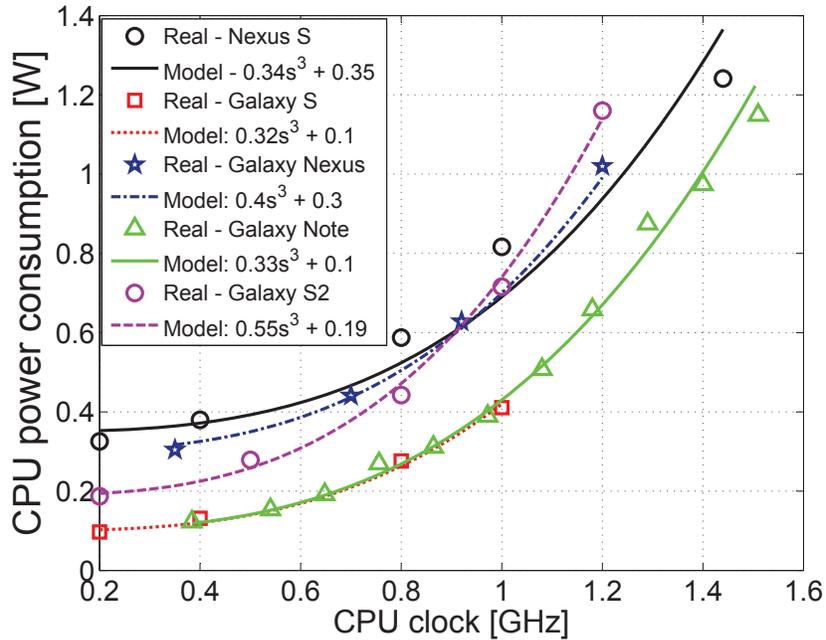
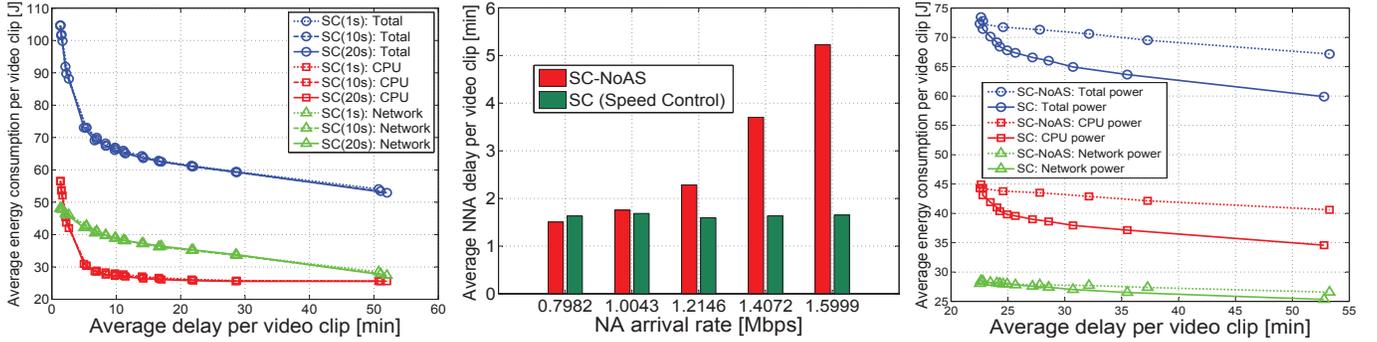


Fig. 3: CPU power consumption vs. CPU speed

is made by us for transmitting dummy data to the server. We make throughput measuring application 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 consumption in W/bit of WiFi and 3G of two devices are comparable with CPU power consumption in W/bit of two devices (CPU power in W/bit assuming that CPU speed is 1GHz and processing density is 1000cycles/bit:  $816.5 \times 10^{-9}$ W/bit (Nexus S),  $688.1 \times 10^{-9}$ W/bit (Galaxy Nexus)). It supports the fact that CPU power is no less important than network power to process the same quantity of data in smartphone.

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

TABLE II: Network power consumptions for two smartphones



(a) Impact of time scale difference: WiFi temporal coverage - 50%,  $\gamma_{NNA}=1200\text{cycles/bit}$  (b) Average delay of NNA in network bottleneck case: WiFi temporal coverage - 0%, average network throughput=0.76Mbps,  $\gamma_{NA}=600\text{cycles/bit}$ ,  $\gamma_{NNA}=600\text{cycles/bit}$  (c) Energy-delay tradeoff in network bottleneck case: WiFi temporal coverage - 0%, average network throughput=0.76Mbps,  $\gamma_{NA}=600\text{cycles/bit}$ ,  $\gamma_{NNA}=600\text{cycles/bit}$

Fig. 4: Characteristic evaluation of SpeedControl

**Real Traces.** We collect workload arrivals, processing densities of NA and NNA and uplink throughputs of 3G and WiFi networks. First, we use real YouTube video data size distribution from [21] 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 range of 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 every smartphone. 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 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 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<sup>3</sup>, 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<sup>4</sup> with delayed network selection [9], 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. 4(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 differentiate between NA and NNA in terms of delay, we carry out the simulation in 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. 4(b), 4(c) depict delay performance of NNA and energy-delay tradeoff of SC-NoAS and SpeedControl when network is the bottleneck. In simulation for Fig. 4(b), the arrival rate of NNA is the same (0.8Mbps), but the arrival rate of NA is increased up to twofold. Fig. 4(b) shows

<sup>3</sup>We also use this estimation method in experiment

<sup>4</sup>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.

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 do 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. 4(c).

**Energy-delay tradeoff.** Fig. 5 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 (37). 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 catch up with the delay performance and energy consumption of an optimal algorithm even though SpeedControl is much low complex algorithm than the optimal algorithm. (iii) SpeedControl saves 56%, 52% (in CPU), 30%, 33% (in total) energy than SpeedControl for 10 minutes average delay as compared to Max+SALSA and DVFS+SALSA, respectively. The power saving gains of SpeedControl comes from the fact that the 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. 6 presents total (CPU+network) energy consumption 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 for more than 5 minutes delay. 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 that SpeedControl exactly know when network side needs the workload to transmit whereas DVFS+SALSA and Max+SALSA does not know what happens in network side. (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.

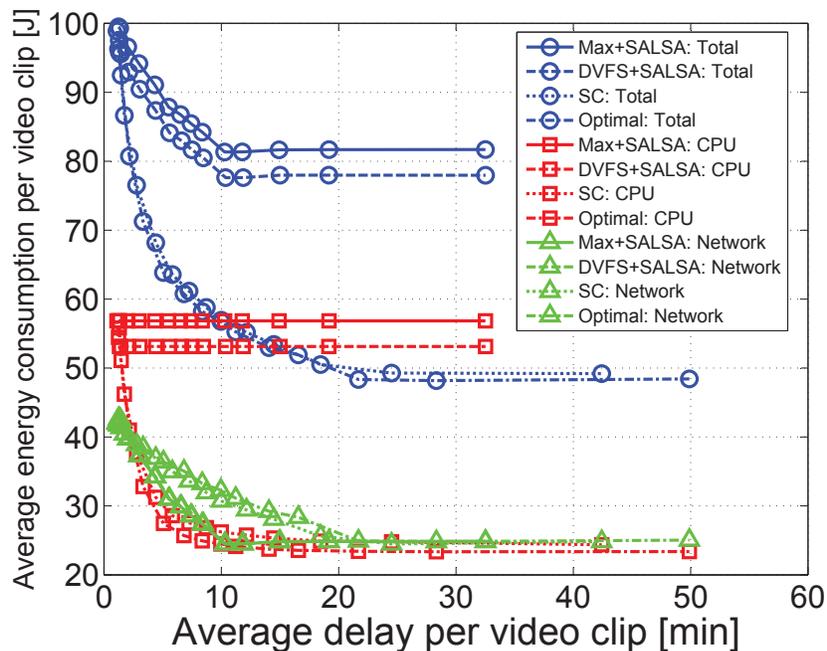


Fig. 5: Energy-delay tradeoff of different algorithms: WiFi temporal coverage - 65%, processing density (NA,NNA) = (300cycles/bit,1200cycles/bit)

(iii) As WiFi temporal coverage is wider, SpeedControl achieves more energy saving for more than 5 minutes delay. 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 relatively more important than networking power. Since our SpeedControl algorithm has great power saving in CPU side whereas networking power consumption is similar with other algorithms like Fig. 5, relative total power saving of SpeedControl is greater than other algorithms.

### 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 [22] which enables to control CPU clock manually. The SpeedControl application determines CPU speed and network selection based on the CPU and network queues and estimation of current uplink throughput. For estimation of uplink throughput, our private server transmits acknowledgement (ACK), which contains uplink throughput information, to device every 5 seconds. Then, the smartphone considers received ACK as the current throughput. If the device does not transmit data in just before time slot, average uplink throughput is considered as current throughput. For experiment, we prepare rooted smartphone (Nexus S) and Monsoon power monitor [20]. A smartphone runs two applications: (i) video encoding application,

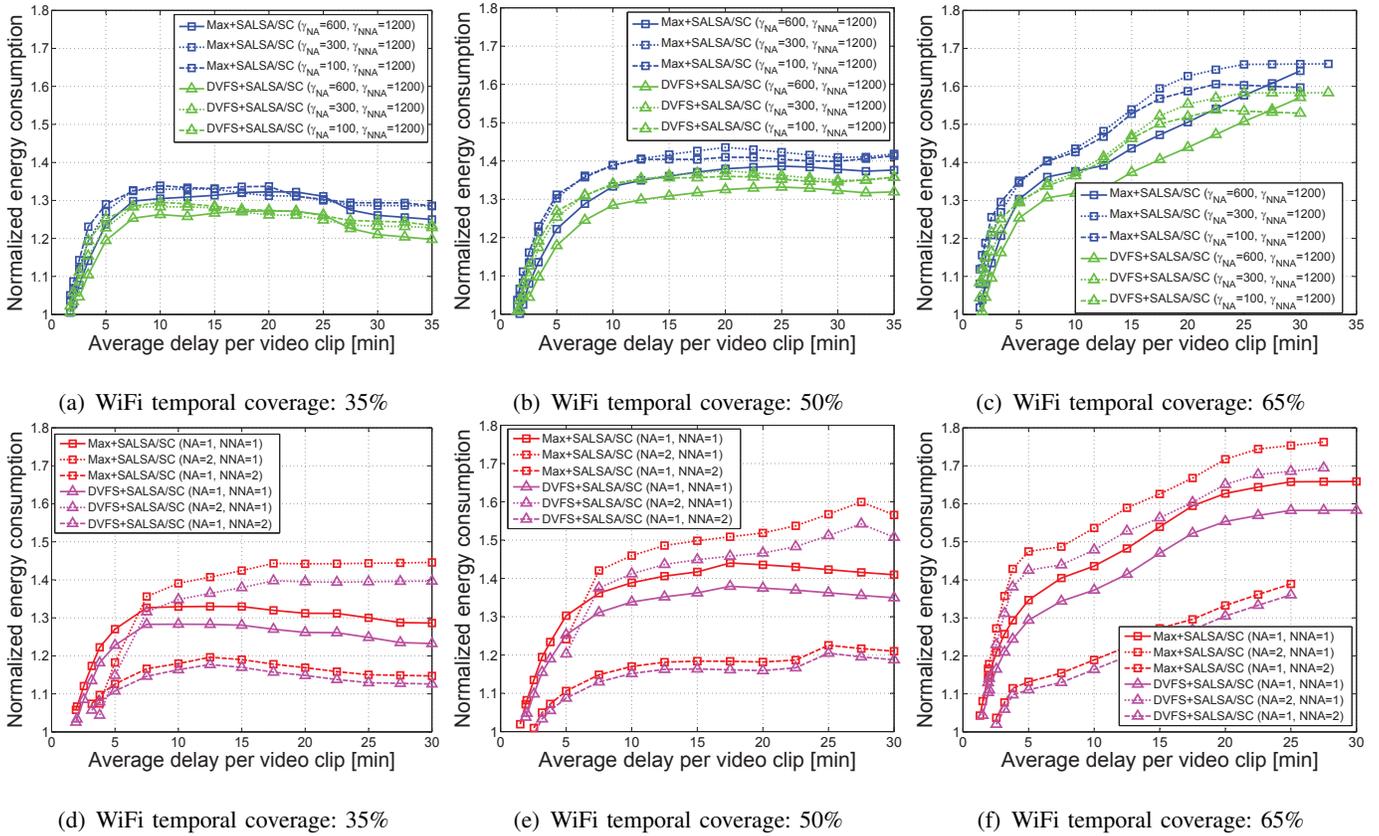


Fig. 6: Normalized energy consumptions for several WiFi temporal coverages: (a)-(c) are the results under the same arrival rate ( $NA:NNA=1:1$ ), (d)-(e) are the results under the same processing density ( $\gamma_{NA} : \gamma_{NNA}=300\text{cycles/bit}:1200\text{cycles/bit}$ )

(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 assume that encoding speed is the same as CPU speed. So, the encoded data is transmitted in the network queue to our private server. We consider 5 video clips (21MByte per one clip) are arrived at the specific time. Also, the smartphone is associated with one WiFi AP<sup>5</sup>, 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 two video clips are fully transmitted.

**Observation.** We obtain three observations from experimental results of several algorithms in Fig. 7. (i) (Fig. 7(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 clip. (ii) (Fig. 7(b)) 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

<sup>5</sup>This WiFi AP is private AP for only one experimental smartphone.

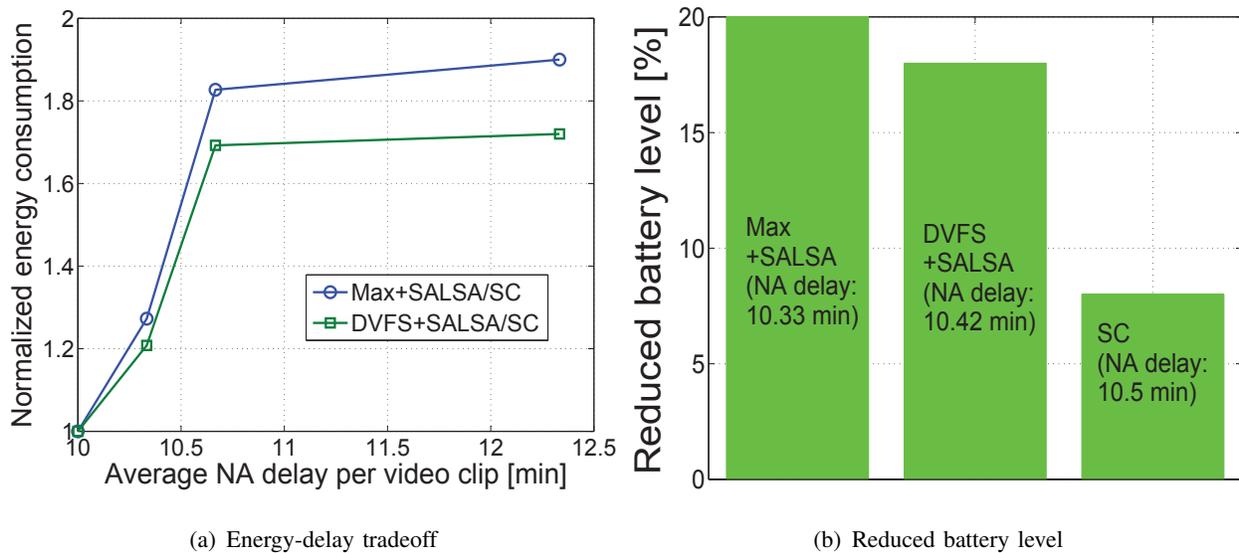


Fig. 7: Experimental results for several algorithms

delay when the starting battery level is 70%. (iii) (Fig. 7(b)) Our SpeedControl consumes 50% and 40% less battery than Max+SALSA and DVFS+SALSA by trading similar delay.

## V. CONCLUSION

In this paper, we seriously investigate key processing and networking features of contemporary smartphones in terms of tradeoff between energy consumption and application 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 when applications can tolerate a certain delay. Also, SpeedControl well isolate 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 such as smartphone power consumption or network states, 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. APPENDIX

### A. Proof of Lemma 1.

*Proof:* From the queueing models of networking application (NA),

$$Q_{NA}^c(t+1) + Q^n(t+1) = [Q_{NA}^c(t) + Q^n(t) - \mu'_l(t) + A_{NA}(t)]^+ , \quad (38)$$

where  $\mu'_i(t) = \min\{\mu_i(t), Q^n(t) + \frac{\theta(t)s(t)}{\gamma_{NA}}\}$  since the actual service quantity of network part is bounded by actual queue length at time slot  $t$ . By taking square on the (38) and using the fact that  $([X]^+)^2 \leq X^2$ , we have:

$$\begin{aligned}
(Q_{NA}^c(t+1) + Q^n(t+1))^2 &\leq (Q_{NA}^c(t) + Q^n(t) - \mu'_i(t))^2 + 2A_{NA}(t)[Q_{NA}^c(t) + Q^n(t) - \mu'_i(t)] + A_{NA}(t)^2 = \\
(Q_{NA}^c(t) + Q^n(t))^2 - 2\mu'_i(t)(Q_{NA}^c(t) + Q^n(t)) + \mu'_i(t)^2 + 2A_{NA}(t)[Q_{NA}^c(t) + Q^n(t) - \mu'_i(t)] + A_{NA}(t)^2 \\
&\leq (Q_{NA}^c(t) + Q^n(t))^2 - 2\mu'_i(t)(Q_{NA}^c(t) + Q^n(t)) + \mu'_i(t)^2 + 2A_{NA}(t)(Q_{NA}^c(t) + Q^n(t)) + A_{NA}(t)^2 \\
&\leq (Q_{NA}^c(t) + Q^n(t))^2 - 2(\mu'_i(t) - A_{NA}(t))(Q_{NA}^c(t) + Q^n(t)) + \mu_{max}^2 + A_{NA,max}^2
\end{aligned} \tag{39}$$

Then, by arranging (39), we have:

$$\begin{aligned}
(Q_{NA}^c(t+1) + Q^n(t+1))^2 - (Q_{NA}^c(t) + Q^n(t))^2 &\leq \\
- 2(\mu'_i(t) - A_{NA}(t))(Q_{NA}^c(t) + Q^n(t)) + \mu_{max}^2 + A_{NA,max}^2
\end{aligned} \tag{40}$$

Similarly, we obtain the following by repeating for non-networking application (NNA).

$$\begin{aligned}
Q_{NNA}^c(t+1)^2 - Q_{NNA}^c(t)^2 &\leq \\
- 2\left(\frac{(1-\theta(t))s(t)}{\gamma_{NNA}} - A_{NNA}(t)\right)Q_{NNA}^c(t) + \frac{s_{max}^2}{\gamma_{NNA}^2} + A_{NNA,max}^2
\end{aligned} \tag{41}$$

Using the above upper bounds of NA and NNA parts, we can obtain the upper bound of following Lyapunov drift plus penalty function.

$$\begin{aligned}
\Delta(L(t)) + V\mathbb{E}\{P^c(s(t)) + P^n(l(t))|\mathbf{Q}(t)\} \\
\leq B + V\mathbb{E}\{P^c(s(t)) + P^n(l(t))|\mathbf{Q}(t)\} - \mathbb{E}\left\{\left(\min\{\mu_i(t), \frac{\theta(t)s(t)}{\gamma_{NA}} + Q^n(t)\} - A_{NA}(t)\right)(Q_{NA}^c(t) + Q^n(t))|\mathbf{Q}(t)\right\} \\
- \mathbb{E}\left\{\left(\frac{(1-\theta(t))s(t)}{\gamma_{NNA}} - A_{NNA}(t)\right)Q_{NNA}^c(t)|\mathbf{Q}(t)\right\}
\end{aligned} \tag{42}$$

where  $B = \frac{1}{2}\left(\frac{s_{max}^2}{\gamma_{NNA}^2} + \mu_{max}^2 + A_{NA,max}^2 + A_{NNA,max}^2\right)$ . This completes the proof.  $\blacksquare$

## B. Proof of Theorem 2.

*Proof:* First, we prove average queue bound. Since  $\lambda_{NA} + 2\epsilon' \in \Lambda_{NA}$  and  $\lambda_{NNA} + \epsilon'' \in \Lambda_{NNA}$ , following relationships can be shown using Theorem 1 that there exists a stationary and randomized policy  $\pi^{*}$ .

$$\mathbb{E}\{P^{\pi^{*}}(t)\} = P^*(2\epsilon' + \epsilon'') \tag{43}$$

$$\mathbb{E}\{A_{NA}(t)\} = \mathbb{E}\left\{\frac{\theta(t)^{\pi^{*}} s(t)^{\pi^{*}}}{\gamma_{NA}}\right\} + \epsilon' \tag{44}$$

$$\mathbb{E}\{A_{NNA}(t)\} = \mathbb{E}\left\{\frac{(1-\theta(t))^{\pi^{*}} s(t)^{\pi^{*}}}{\gamma_{NNA}}\right\} + \epsilon'' \tag{45}$$

$$\mathbb{E}\left\{\frac{\theta(t)^{\pi^{*}} s(t)^{\pi^{*}}}{\gamma_{NA}}\right\} = \mathbb{E}\{\mu_i(t)^{\pi^{*}}\} + \epsilon' \tag{46}$$

where  $P^*(2\epsilon' + \epsilon'') = P^{c*}(\lambda_{NA} + \lambda_{NNA} + 2\epsilon' + \epsilon'') + P^{n*}(\lambda_{NA} + 2\epsilon')$  is the optimal average sum of CPU and network power for the average arrival rate  $\lambda_{NA} + \lambda_{NNA} + 2\epsilon' + \epsilon''$ . By applying the above equations (43), (44), (45) and (46) in Lemma 1, we obtain following:

$$\begin{aligned} & \Delta(L(t)) + V\mathbb{E}\{P^{c,OPT}(s(t)) + P^{n,OPT}(l(t))\} \\ & \leq B + VP^*(2\epsilon' + \epsilon'') - \mathbb{E}\{2\epsilon'(Q_{NA}^c(t) + Q^n(t))|\mathbf{Q}(t)\} - \mathbb{E}\{\epsilon''Q_{NNA}^c(t)|\mathbf{Q}(t)\} \end{aligned} \quad (47)$$

By taking expectations on both NA and NNA side queues, and using the fact that  $P^*(2\epsilon' + \epsilon'') \leq P_{max}^c + P_{max}^n = P^{max}$ ,

$$\mathbb{E}\{L(t+1) - L(t)\} + 2\epsilon'\mathbb{E}\{(Q_{NA}^c(t) + Q^n(t))\} + \epsilon''\mathbb{E}\{Q_{NNA}^c(t)\} \leq B + VP_{max}^c \quad (48)$$

By taking  $2\epsilon' = \epsilon'' = \epsilon$  and dividing both sides of (48) with  $\epsilon$  and summing over  $t = \{0, 1, \dots, T-1\}$ , we obtain following:

$$\mathbb{E}\{L(T) - L(0)\} + \sum_{t=0}^{T-1} \mathbb{E}\{Q_{NA}^c(t) + Q_{NNA}^c(t) + Q^n(t)\} \leq \frac{(B + VP_{max}^c)T}{\epsilon} \quad (49)$$

By dividing both sides of (49) with  $T$  and using the fact that  $L(t) \geq 0$  and taking as  $T \rightarrow \infty$ , this completes the proof of NA and NNA queue bounds. Next, we prove average power bound. By the fact that  $\epsilon(Q_{NA}^c(t) + Q_{NNA}^c(t) + Q^n(t)) \geq 0$ , we obtain following inequality from (47):

$$\Delta(L(t)) + V\mathbb{E}\{P^{c,OPT}(t) + P^{n,OPT}(t)|\mathbf{Q}(t)\} \leq B + VP^*(2\epsilon' + \epsilon'') \quad (50)$$

where  $P^{c,OPT}(t)$  and  $P^{n,OPT}(t)$  are CPU and network power consumption when an optimal algorithm is adopted. By summing (50) over  $t = \{0, 1, \dots, T-1\}$  and using the fact that  $L(t) \geq 0$ ,  $Q_{NA}^c(0) = 0$ ,  $Q_{NNA}^c(0) = 0$ ,  $Q^n(0) = 0$ , and dividing  $T$ , we have:

$$\frac{1}{T} \sum_{t=0}^{T-1} \mathbb{E}\{P^{c,OPT}(t) + P^{n,OPT}(t)\} \leq \frac{B}{V} + P^*(2\epsilon) \quad (51)$$

Then, taking as  $T \rightarrow \infty$  and using the Lebesgue's dominated convergence theorem, this completes the proof of average power bound. ■

## 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] 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.
- [7] 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.
- [8] 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.
- [9] 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.
- [10] 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.
- [11] "Google Play." [Online]. Available: [https://play.google.com/store/apps/collection/topselling\\_free](https://play.google.com/store/apps/collection/topselling_free)
- [12] "Operating Systems, iOS 6.0." [Online]. Available: <http://www.apple.com/ios/>
- [13] "Operating Systems, Android." [Online]. Available: <http://www.android.com/>
- [14] M. Neely, "Stochastic network optimization with application to communication and queueing systems," *Synthesis Lectures on Communication Networks*, pp. 1–211, 2010.
- [15] "Kernel governors, modules, I/O scheduler, CPU tweaks AIO app configs." [Online]. Available: <http://forum.xda-developers.com/showthread.php?t=1369817>
- [16] 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.
- [17] 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.
- [18] 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.
- [19] 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.
- [20] "Power Meter Device: Monsoon Power Monitor." [Online]. Available: <http://www.msoon.com/LabEquipment/PowerMonitor/>
- [21] "Dataset for statistics and social network of YouTube videos." [Online]. Available: <http://netsg.cs.sfu.ca/youtubedata/>
- [22] "NSTools application and open source code v1.16." [Online]. Available: <http://forum.xda-developers.com/showthread.php?t=1333696>