Energy-Aware Web Browsing in 3G Based Smartphones

Bo Zhao, Qiang Zheng, Guohong Cao
The Pennsylvania State University
Email: {bzhao,quz103,gcao}@cse.psu.edu

Sateesh Addepalli
Cisco Systems
Email: sateeshk@cisco.com

***Abstract*—Smartphone based web browsing wastes a lot of power when downloading webpages due to the special characteristics of the 3G radio interface. In this paper, we identify these special characteristics, and address power consumption issues through two novel techniques. First, we reorganize the computation sequence of the web browser when loading a webpage, so that the web browser can first run the computations that will generate new data transmissions and retrieve these data from the web server. Then, the web browser can put the 3G radio interface into low power state, release the radio resource, and then run the remaining computations. Second, we introduce a practical data mining based method to predict the user reading time of webpages, based on which the smartphone can switch to low power state when the reading time is longer than a threshold. To demonstrate the effectiveness of our energy-aware approaches, we develop a testbed with Android phones on T-Mobile UMTS network. Experimental results show that our approach can reduce the power consumption of smartphone by more than 30% during web browsing, and reduce the webpage loading time by 17%.**

I. Introduction

Web browsing is one of the most important and commonly used service provided by smartphones. However, the current smartphone web browser wastes a lot of power when downloading webpages. There have been lots of research on optimizing the power usage of smartphones [1], [2], but they only focus on reducing the power consumption of one component such as display. Although some of them also consider the power consumption of the wireless interface [3], most of them focus on the WiFi interface which has different characteristics from the 3G wireless interface which consumes much more power.

In UMTS 3G networks, multiple timers are used to control the radio resource, and the timeout value for releasing the resource can be more than 15 seconds [4]. Thus, it is possible that the wireless radio interface continues to consume a large amount of power before the timer times out, even when there is no network traffic. One advantage of this approach is that it can reduce the latency of next possible data transmission that arrives before the timer expires, because the connection between the smartphone and the backbone network is still available. Otherwise, the backbone network has to allocate the radio resource again, which will consume more time and power. As a result, simply adjusting the timer may not be a good solution for saving power.

Due to the limited computation capability, when opening a webpage, current smartphone web browser takes a long time for downloading and processing all objects (e.g., javascripts) of the webpage. As a result, the data transmissions are distributed along the whole webpage downloading duration, and then the data rate at any instant time is quite low. Although there are many idle times between these data transmissions, each idle time is still smaller than the time out value, and these data transmissions reset the timers again and again before they expire. Consequently, the radio interface is always on and the radio resource cannot be released, which consumes lots of power and decreases the network capacity.

In this paper, we address the power consumption issue in smartphone based web browsing through two novel techniques. First, we reorganize the computation sequence of the web browser when loading a webpage. There are various computations when loading a webpage such as HTML parsing, JavaScript code execution, image decoding, style formatting, page layout, etc. These computations generally belong to two categories based on whether they will generate new data transmissions from the web server. We want to separate these two types of computations so that the web browser can first run the computations that will generate new data transmissions and retrieve these data. Then, the web browser can put the 3G radio interface into low power state, release the radio resource, and then run the remaining computations which may take 40–70% of the processing time for loading webpages [5]. Thus, a significant amount of power and radio resource can be saved.

The aforementioned technique can be applied to webpages which require a long processing time. For webpages that have short processing time, we propose another novel technique. The idea is to predict the user reading time on the webpage after it is downloaded. If this time is longer than a threshold, the radio interface can be put into low power state. Since smartphones have limited computation capability, we propose a low overhead prediction algorithm based on Gradient Boosted Regression Trees (GBRT) [6].

To demonstrate the effectiveness of our energy-aware approaches, we implement a prototype based on Android phones on T-Mobile UMTS network. We collect real data traces to evaluate our energy-aware approaches. Experimen-

This work was supported in part by the US National Science Foundation (NSF) under grant number CNS-1218597, and by Network Science CTA under grant W911NF-09-2-0053.

tal results show that our approach can reduce the power consumption of smartphone by more than 30% during web browsing, and reduce the webpage loading time by 17%. More specifically, the contributions of this paper are as follows.

- To the best of our knowledge, we are the first to achieve power saving for web browsing on smartphones by reorganizing the computation sequence of web browser. Moreover, this approach can increase the network capacity without any extra equipment and it does not modify the communication protocol between the smartphone and the backbone network.
- The proposed light-weight GBRT prediction method has high accuracy and can be used to further save power during user reading time.
- We implement the prototype and collect real traces which are used to validate the effectiveness of the proposed energy-efficient approaches.

The rest of this paper is organized as follows. Section II and Section III introduce the background and motivation of our work, respectively. Section IV presents the two energy saving techniques. Section V presents our test-bed and experimental results. Section VI discusses related work, and Section VII concludes the paper.

II. Background

In this section, we introduce the background knowledge related to power consumption of the 3G radio interface and the design of smartphone web browser.

A. Power Consumption of the 3G Radio Interface

To efficiently utilize the limited radio resource of the backbone network [7], the 3G Radio Resource Control protocol defines the following three states for smartphones to control their radio interfaces.

- IDLE state: the smartphone does not have any signaling connection with the backbone network, and hence it cannot transmit user data. The radio interface of the smartphone in IDLE state consumes very little power as shown in Fig. 1.
- DCH state: the backbone network allocates dedicated transmission channels (uplink and downlink) to the smartphone, so that the smartphone can transmit user data and signaling information at high speed. This requires high level of power as shown in Fig. 1.
- FACH state: smartphone in this state has no dedicated transmission channel. Hence, it can only transmit user data and signaling information through common shared transmission channels at low speed (up to a few hundred bytes/second). As shown in Fig. 1, data transmission in the FACH state requires about half of the power in the DCH state.

When a smartphone in IDLE wants to transmit user data, it has to switch to DCH by first establishing a signaling connection with the backbone network, and then obtaining dedicated transmission channels. This process requires ten of control message exchanges between the smartphone and the backbone network, which takes more than one second.

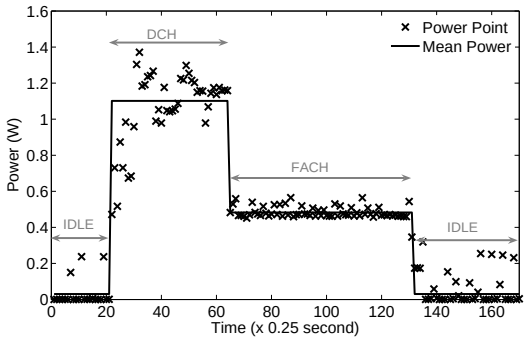


Figure 1. The power level of the 3G radio interface on smartphones at different states.

The backbone network uses a timer to determine when to release the dedicated transmission channels allocated to the smartphone. The timer ($T1$) is usually set to 4 seconds. The backbone network triggers $T1$ when each data transmission is complete and resets it whenever a new data transmission happens before it expires. When $T1$ expires, the dedicated transmission channels allocated to the smartphone are released, and the smartphone switches to FACH.

After the smartphone switches to FACH, the backbone network triggers another timer $T2$ to determine when to release the signaling connection, which is usually set to 15 seconds [4]. Before $T2$ expires, the signaling connection between the smartphone and the backbone network still exists. If the backbone network or the smartphone needs to transmit user data, the backbone network allocates dedicated transmission channels to the smartphone and the smartphone switches to DCH. Since the signaling connection still exists, the latency of switching the smartphone from FACH to DCH is smaller than that of switching it from IDLE to DCH. When $T2$ expires, the signaling connection is released and the smartphone enters IDLE.

B. Web Browser Design

Today's web browser is more complex since it has to process various script code such as JavaScript embedded in HTML documents. Further, it has to process Cascading Style Sheets (CSS), which is used to describe the presentation semantics and the style rules of a webpage such as layout, color and fonts. Document Object Model (DOM) is an interface that allows programs and scripts to update the content, structure and style of HTML documents. After the HTML code has been parsed, the nodes in the DOM tree store the HTML data. After the CSS code has been parsed, the style and layout properties are assigned to these nodes in the DOM tree. Then the web browser can display them on the screen.

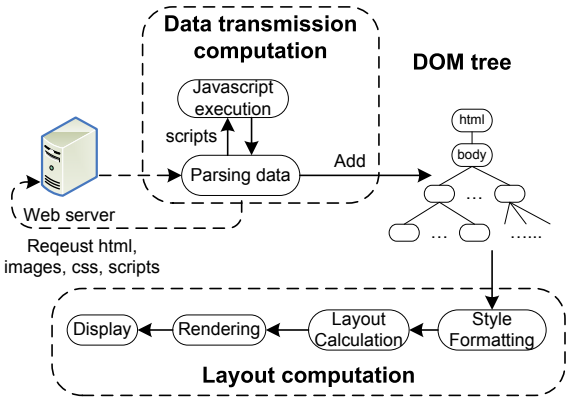


Figure 2. The workflow of webpage processing in smartphone based web browser.

After the web browser gets the main HTML page, the data transmission mainly comes from three kinds of sources: HTML, JavaScript, and CSS. In HTML and CSS, content objects such as HTML files, JavaScript files, images, and flash objects, are referenced by URLs. Hence, the web browser needs to fetch and add them to the DOM tree. For JavaScript code, it is either transformed to HTML code and then fetches content objects, or it can directly get content objects from the web server.

Fig. 2 shows a simplified workflow of webpage processing in smartphone web browsers. Loading HTML page triggers a set of events. First, the HTML code is parsed. If there is JavaScript code, it is delivered to the interpreter engine to process. If it includes URLs for objects like images, html files, and flash objects, these objects are requested. Each object is added to the DOM tree as a node. Next, to provide better user experience, the web browser processes the layout information such as image decoding, style formatting, page layout calculation, and page rendering. Then it can draw a partially rendered display on the screen before finishing loading the whole webpage [5].

As shown in Fig. 2, in current smartphone web browser, there are two types of computations associated with each incoming object. The first type is the computation that generates new data transmissions such as HTML and CSS file parsing and JavaScript code execution, which is referred to as the *data transmission computation*. The second type is the computation that does not cause data transmission. This type of computation is used to lay out the webpage such as image decoding, style formatting, page layout calculation and page rendering, which is referred to as the *layout computation*.

III. Motivation

In this section, we introduce how to save power for smartphones in the DCH state and FACH state, respectively.

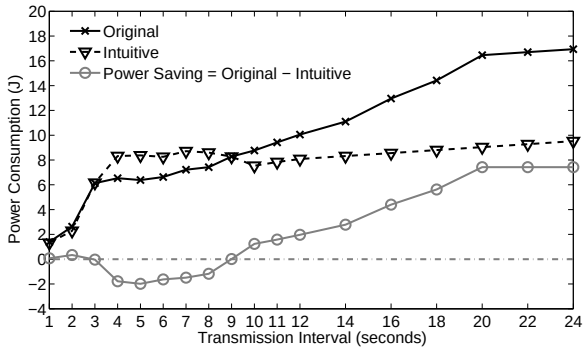


Figure 3. The power consumption with different transmission intervals.

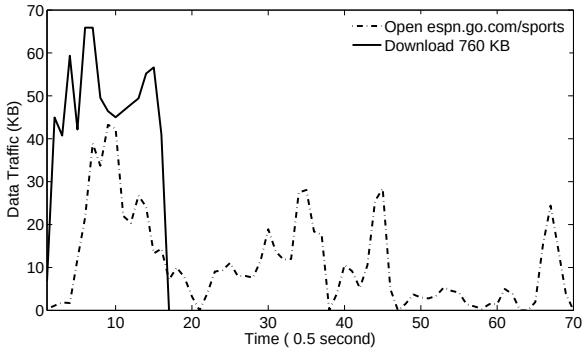


Figure 4. The traffic load of web browsing and data downloading

A. Power Saving in the DCH State

Since smartphone in the DCH state consumes much more power than in IDLE, an intuitive approach for saving power is to switch the smartphone from DCH to IDLE immediately after each data transmission. However, this approach does not work if the next data transmission happens quickly. First, reestablishing the signaling connection and switching the smartphone from IDLE to DCH needs to consume power. The power consumed in these two steps may be more than the power saved by switching to IDLE, if the time interval between the two data transmissions is short. Second, reestablishing the signaling connection increases the delay of data transmission, and then it affects the user experience of web browsing.

We measure the delay and power consumption of this intuitive approach through a simple experiment. We send 1 KB data from an Android smartphone to a remote server. After receiving 1 KB data through 3G network, the smartphone immediately switches to IDLE until another data transmission. This approach causes about 1.75 seconds extra delay compared with the original approach. We also measure the power saving of this intuitive approach with different data transmission intervals, which is shown in Fig. 3. This intuitive approach can save power only when the data transmission interval is larger than 9 seconds. Otherwise, it consumes more power than the original approach. For most webpages, the data transmission interval during webpage

loading is smaller than 4 seconds. As a result, this intuitive approach consumes more power.

Although the web browser takes a long time to get the objects of a webpage, the data transmissions are distributed along the whole webpage loading time and the traffic rate is quite low. Fig. 4 shows the traffic load of opening webpage espn.go.com/sports/, where the smartphone web browser takes 34.5 seconds to download all 760 KB data. The data is downloaded through several data transmissions which are spread along the whole webpage loading time. We open a socket client to download the same amount of data (760 KB), and it only takes 8 seconds as shown in the figure.

If we can group all data transmissions together, the objects of a webpage can be downloaded in a shorter time. Then, we can switch the smartphone from DCH to FACH (and then to IDLE), because there will be no data transmission for the current webpage. Hence, power can be saved via reducing the time in the DCH state. The details on how to reorganizing the computation sequence will be presented in Section IV-A.

B. Power Saving in the FACH State

As introduced in Section II-A, timer $T2$ controls how long the smartphone stays in the FACH state, which is usually 15 seconds. Since most users spend more than 20 seconds on reading downloaded webpages and there is no data transmission for 80% webpages [8] during this reading time, keeping smartphone in FACH only wastes power. Thus, we should switch the smartphone from FACH to IDLE to save power.

Our approach for saving power in the FACH state is based on predicting the webpage reading time. If the reading time is larger than a threshold, the smartphone will switch from FACH to IDLE. Since smartphone has limited computation capability, we cannot use complex prediction algorithms, and we will introduce our prediction algorithm in Section IV-C.

IV. System Design

A. Reorganizing the Computation Sequence

We reorganize the computation sequence of the web browser to retrieve all data in the webpage as fast as possible. In the original web browser design, the data transmission computation and the layout computation are mixed, as shown in Fig. 2. In this subsection, we explain how to separate them to save power.

Fig. 5 compares the computation sequence of our energy-aware approach with that of the original web browser for opening a webpage. At time slot 1, the original web browser spends its computation resource on both data transmission and layout computation. It processes one object and adds it to the DOM tree. In our approach, the browser focuses on data transmission and ignores layout. Thus, our approach can process more objects and add them to the DOM tree at time slot 1. At time slot 2, our approach processes all the

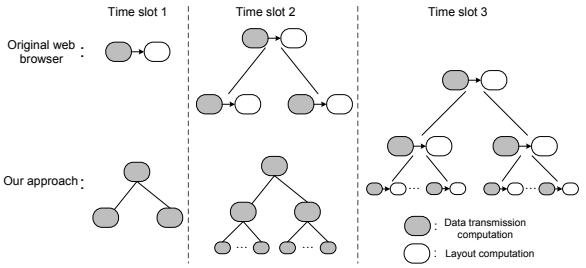


Figure 5. The computation sequence for opening webpage and building the DOM tree

objects and builds the complete DOM tree. At time slot 3, no data transmission will happen, and our approach focuses on computing the page layout. Although the original web browser can display partial webpage content on the screen at each time slot, it keeps generating data transmission until time slot 3. Finally, both approaches have the same complete DOM tree and display the same webpage although our approach may be a little bit faster in practice.

Next, we describe how to separate different type of objects in detail. For computations related to HTML files, we scan them to fetch the objects (images, JavaScript files, flash objects, and CSS files) referred by URLs. If the JavaScript and HTML files do not request any object, the data transmission is finished immediately after downloading them. Next, the HTML files are parsed to build the DOM tree for grammar correction, because the JavaScript code may use the HTML element in the DOM tree when being processed by the interpreter engine.

For computations related to CSS code and files, we only scan them to fetch the objects (images and CSS files) referred by URLs, but do not parse them. Thus, the web browser does not spend any computation on parsing them and generating the style rules. Image files and other multimedia objects can be saved in memory instead of being delivered to the web browser. Hence, the web browser does not spend any computation time on decoding the multimedia objects.

Separating the computation for JavaScript files or the Java-Script code in HTML file is the most difficult task. Generally, JavaScript codes are distributed in several JavaScript files and HTML files. Since they are executed in global context, we have to run them in sequence to check if they will generate any data transmission or not. Besides, JavaScript codes are much more complex than HTML or CSS codes, and there is no simple approach to find out if they will generate new data transmission without executing them.

After all HTML and JavaScript files are processed and all the CSS files are scanned, all data transmission computations are done. Next, we process the layout computation (e.g., image decoding, CSS files processing, style formatting, layout calculation, and rendering) which does not cause data transmission.

B. Intermediate Display

To improve user experience, original web browsers always draw intermediate display and update it frequently when loading webpages. However, this approach has two disadvantages. First, although the web browser already has the web content, it has to wait before displaying the intermediate results. This is because the browser needs to associate DOM nodes and CSS style rules to lay out these data. However, since the CSS file is large and complex, it takes a lot of processing time to extract the rules [5]. Another disadvantage is that the browser wastes a lot of computation resource to frequently redraw and reflow the intermediate display before the final display.

By reorganizing the computation sequence, our approach draws the final display at the end of webpage loading, and thus it saves the computation on redrawing or reflowing intermediate display. To improve user experience, we present a low-overhead approach to draw intermediate display with little layout computation during the data transmission time. When the browser begins to open a webpage, it first downloads the webpage which has a lot of text content and the basic html format information. After parsing 1/3 webpage content, we can draw a simplified intermediate display. Because this display does not need CSS rules, style format, or images, it can be displayed much earlier than the original web browser. Moreover, because the interval between the first intermediate display and the final display is only around $10 \sim 20$ seconds, we do not update the intermediate display during the loading time to avoid unnecessary computation on redraw and reflow. For webpages with mobile version, since the interval is very short ($1 \sim 2$ seconds) between intermediate display and final display, our approach only draws the final display.

C. Predicting the Reading Time

In this subsection, we present a machine learning based approach to predict the user reading time, based on which we can decide if the smartphone should switch to IDLE.

1) Gradient Boosted Regression Trees: Gradient Boosted Regression Tree (GBRT) [6] is a technique of predictive data mining, which is used for constructing statistical *decision tree* models from historical data. *Decision tree* is a decision support tool that uses a tree-like graph or model of decisions and their possible consequences. These models are used to predict the value of future data. In the "predictive learning" problem, one has a system consisting of a "output" variable y and a set of "input" variables $\mathbf{x} = \{x_1, \cdots, x_n\}$. Using a "training" sample $\{y_i, \mathbf{x_i}\}_1^N$ of known $(y, \mathbf{x})$ values, the goal is to obtain an estimate or approximation $F(\mathbf{x})$ of the function $F^*(x)$ mapping $\mathbf{x}$ to y, which minimizes the expected value of some specified loss function $L(y, F(\mathbf{x}))$ over the joint distribution of all $(y, \mathbf{x})$ values.

Table I
FEATURE DETAILS

Feature	Description
Reading Time	The duration from the webpage is completely opened to the time when the user clicks to open another webpage
Transmission Time	Data transmission time
Webpage Size	The data size of the webpage without considering the figures
Download Objects	The number of total downloaded objects
Download JavaScript files	The number of downloaded JavaScript files
Download Figures	The number of downloaded figures
Figure Size	Size of the total downloaded figures
JavaScript Running Time	The time for processing all the JavaScript code
Second URL	The number of secondary URLs
Page Height	The height of the webpage
Page Width	The width of the webpage

$$F^*(\mathbf{x}) = \arg\min_F E_{y,\mathbf{x}} L(y, F(\mathbf{x})) \quad (1)$$

$$= \arg\min_F E_{\mathbf{x}}[E_y(L(y, F(\mathbf{x})))|\mathbf{x}] \quad (2)$$

The GBRT method assumes a real-valued y and seeks an approximation $F(x)$ in the form of a weighted sum of functions $h_i(x)$ from some class $\mathcal{H}$ called based learners:

$$F(\mathbf{x}) = \sum_{i=1}^{M} \beta_i h_i(\mathbf{x}) \quad (3)$$

Gradient Boosting: In accordance with the empirical risk minimization principle, the method tries to find an approximation $F(\mathbf{x})$ that minimizes the average value of the loss function on the training set. This is achieved by starting with a model consisting of a constant function $F_0(x)$ and then incrementally expanding it in a greedy fashion as follows.

$$F_0(\mathbf{x}) = \arg\min_\beta \sum_{i=1}^{n} L(y_i, \beta) \quad (4)$$

$$F_m(\mathbf{x}) = F_{m-1}(\mathbf{x}) + \beta_m h_m(\mathbf{x}) \quad (5)$$

$$\beta_m = \arg\min_\beta \sum_{i=1}^{n} L(y_i, F_{m-1}(x_i) + \beta h_m(x_i)) \quad (6)$$

Where F is restricted to be a function from the class $\mathcal{H}$ of base learner functions. Eq.(4) and Eq.(5) are called "gradient boosting".

Gradient Tree Boosting: Here we consider the special case where each base learner is a J-terminal node decision tree. Each decision tree model itself has the additive form:

$$h(\mathbf{x}; \{b_j, R_j\}_1^J) = \sum_{j=1}^{J} b_j 1(\mathbf{x} \in R_j) \quad (7)$$

Here $\{R_j\}_1^J$ represents disjoint regions that collectively cover the space of all joint values of the predictor variables $\mathbf{x}$. These regions are represented by the terminal nodes of the corresponding decision tree. The indicator function $1(.)$ has value one if its argument is true, and zero otherwise. The "parameters" of this base learner Eq.(7) are the coefficients $\{b_j\}_1^J$, and the quantities that define the boundaries of the regions $\{R_j\}_1^J$. These are the splitting variables, and the values of those variables, that represent the splits at the nonterminal nodes of the decision tree. Because the regions are disjoint, Eq.(7) is equivalent to the prediction rule: if $\mathbf{x} \in R_j$ then $h(\mathbf{x}) = b_j$. Then for a decision tree:

$$F_m(\mathbf{x}) = F_{m-1}(\mathbf{x}) + \beta_m \sum_{j=1}^{J} b_{jm} 1(\mathbf{x} \in R_j m) \quad (8)$$

Here $\{R_{jm}\}_1^J$ are the regions defined by the terminal nodes of the decision tree at the mth iteration. They are constructed to predict $\{y_i\}_1^N$ by least squares. $\{b_{jm}\}$ presents the corresponding least-square coefficient. m is the number of decision trees. J is the number of terminal nodes in each decision tree.

2) Setup the Trace Collection Program: We modify the Android web browser to collect 10 features of the webpage when opening it, which are listed in Table I. We use feature "Reading Time" to verify the accuracy of output y from GBRT and use the remaining features as input predictors $\mathbf{x} = \{x_1, \cdots, x_{10}\}$ to the GBRT.

3) Training the Model: To train the model, we use square error as the employed loss functions $L(y, F) = (y - F)^2$. This leads to the following generic Algorithm 1.

Algorithm 1 : Regression Tree Boost

Get the input training set $(x_i, y_i)_{i=1}^n$
Number of iteration M.
Initialize the model with a constant value:
$F_0(\mathbf{x}) = median\{y_i\}_1^N$
for $m = 1$ to M **do**
$\tilde{y}_i = -\left[\frac{\partial L(y_i, F_{m-1}(\mathbf{x_i}))}{\partial F_{m-1}(\mathbf{x_i})}\right]_{F(\mathbf{x})=F_{m-1}(\mathbf{x})}, i = 1, \ldots, N$
$\{R_{jm}\}_1^J = J - terminalnodetree(\{\tilde{y}_i, \mathbf{x_i}\}_1^N)$
$\gamma_{jm} = \arg\min_\gamma \sum_{\mathbf{x_i} \in R_{jm}} L(y_i, F_{m-1}(\mathbf{x_i}) + \gamma)$
$F_m(\mathbf{x}) = F_{m-1}(\mathbf{x}) + \sum_{j=1}^{J} \gamma_{jm} 1(\mathbf{x} \in R_{jm})$
end for

Due to the high computation overhead, the model is trained either offline on a PC or on the smartphone when it is connected to a power source. Then, we deploy the tree model to the prediction program which is embedded in the web browser program. When the web browser finishes parsing a webpage, this program will put feature data $\mathbf{x} = \{x_1, \cdots, x_{10}\}$ to the tree model to predict the reading time. Since the model is composed of decision trees, the prediction workload is very low and it can be run on mobile devices. In addition, we do not need to frequently update the tree model, because the user behavior of web browsing generally does not change too much.

4) Interest Threshold: Based on our collected traces (details in Section V-D1), the reading time of a webpage is related to both webpage features and user interests in the webpage. For example, suppose that two news webpages have similar features listed in Table I. One is about games and the other is about finance. A user may spend more time on the webpage about games if he has more interests in game than finance.

To match the user interest, we need to parse the content of a webpage when opening it to collect the word frequency and classify the content based on keywords like "game", "finance", and "weather". Parsing the content and classifying webpages will consume too much power on computation limited smartphones.

Instead of parsing the webpage content, we use a predefined *interest threshold* for prediction. If the reading time of a webpage is larger than the *interest threshold*, it means that the user is interested in this webpage. To improve the prediction accuracy, we only predict the reading time of the webpages that the user is interested in. After a webpage is opened, we will wait until the reading time is larger than the *interest threshold*. Then we can predict and decide if the smartphone should switch to IDLE. In Section V, we will describe how to set the interest threshold in detail.

5) Our Energy-Aware Approach: In this subsection, we present our energy-aware approach as shown in Algorithm 2. The algorithm has two different modes: the delay driven mode which optimizes delay, and the power driven mode which optimizes power. Recall that improperly moving to the IDLE state may increase the power consumption and the data transmission delay as explained in Section III. In delay driven mode, if the predicted reading time (T_r) is shorter than T_d, new data transmission may come during the FACH state, and hence the smartphone will not go to IDLE to avoid increasing the data transmission delay. Here we set T_d to 20 seconds which is T_1 +T_2. In power driven mode, as long as the predicted reading time is longer than T_p, the smartphone will go to IDLE to save power although it may increase the transmission delay in some cases. As shown in Figure 3, T_p = 9 seconds. Table II lists the parameters used in Algorithm 2.

V. Performance Evaluations

In this section, we first describe our experiment setup, and then evaluate the delay and power consumption of web

Table II
PARAMETER DESCRIPTION

Parameters	Description
T_r	Predicted reading time (sec)
α	Interest threshold (sec)
T_d	Time duration threshold ($T_1 + T_2$) for delay driven mode
T_p	Time duration threshold for power driven mode
$Mode$	Power driven or delay driven

Algorithm 2 : Energy-Aware Approach

Begin to open a webpage
Data transmission computation is done.
Layout computation is finished
Collect features $\mathbf{x} = \{x_1, \cdots, x_{10}\}$
Webpage is opened
Wait for α seconds.
Get T_r from the prediction model with $\mathbf{x}$
if ($T_r > T_d$) OR ($T_r > T_p$ AND mode == power) **then**
  switch to IDLE state
end if

browsing.

A. The Experimental Setup

1) The Experimental Equipment: We use Android Phone connecting with the T-Mobile 3G UMTS network as the testbed. To accurately measure the power consumption of the smartphone, we use Agilent E3631A Power Supply to provide the current with constant voltage (3.706 V) to the smartphone instead of using the battery. The Agilent E3631A connects to our laptop through the "NI GPIB-USB-HS" cable. We install LabVIEW on the laptop, and use it to program the Agilent E3631A to capture the current of the smartphone every 0.25 second. Then, we can easily calculate the power consumption of the smartphone. We modify the Android browser to reorganize the data transmission computation and layout computation.

2) Benchmark Webpages: Table III lists all webpages in the benchmark, taken from the most popular webpages from Alexa website [9]. Most webpages of these popular websites have mobile versions for smartphones, while their subpages (lower level webpages) are full versions to show various content. Thus, we separate the benchmark into two parts: the *mobile version benchmark* and the *full version benchmark*. The mobile version benchmark consists of the mobile version of the top popular webpages. Some mobile version webpages have the corresponding full versions that can be displayed on smartphone, such as www.amazon.com. We add them to the full version benchmark. If a mobile version webpage does not have its full version, one of its subpage with full version is added to the full version benchmark.

Table III
BENCHMARK WEBPAGES

Mobile version	Full version
cnn	edition.cnn.com/WORLD/
ebay	www.motors.ebay.com
espn.go.com	espn.go.com/sports
amazon	amazon full version
msn	home.autos.msn.com
myspace	www.myspace.com/music
bbc.co.uk	bbc.com/travel
aol	www.popeater.com/celebrities/
nytime	www.apple.com
youtube	hotjobs.yahoo.com

B. Data Transmission Time

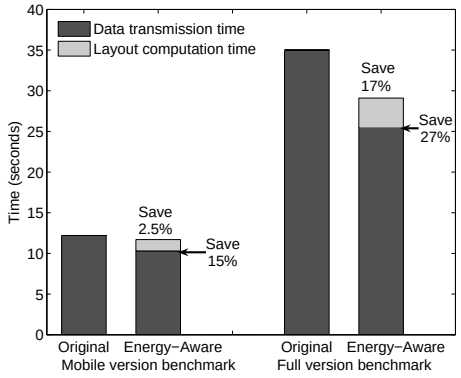
In the original web browsing design, data transmission computation and layout computation are mixed, and hence the data transmission time is the webpage loading time. Our energy-aware design separates these two kinds of computations. Thus, the webpage loading time consists of data transmission time and layout computation time.

Fig. 6(a) shows the average data transmission time on the mobile version benchmark and the full version benchmark. Our approach reduces the data transmission time by 15% and 27% on the mobile version benchmark and full version benchmark, respectively. Our approach also reduces the webpage loading time. For full version benchmark, the loading time is reduced by 17%. The original web browser frequently redraws and reflows intermediate display when loading webpages. Our approach only draws a simple intermediate display and lays out the final display, and hence it can load webpages more quickly. The loading time reduction achieved on the mobile version benchmark is smaller than that on the full version benchmark because most mobile version webpages are quite simple. As a result, the layout computation time is very short and redrawing and reflowing intermediate display do not incur much computation.

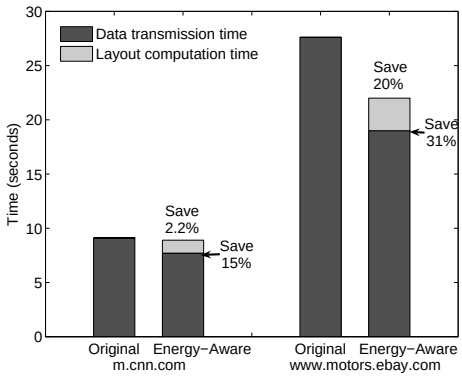
In addition to the average loading time, we also study the loading time of two popular webpages, where m.cnn.com is selected as the mobile version webpage and www.motors.ebay.com represents the full version webpage. As shown in Fig. 6(b), the results have the same trend as those in Fig. 6(a). Our approach can reduce the data transmission time by about 30% for www.motors.ebay.com and 15% for m.cnn.com.

C. Power Consumption

The power consumption of smartphones includes the power consumed for display and system maintenance. Table IV lists the power consumption of the smartphone in different states, where the DCH state consumes the largest amount of power. When the smartphone stays in FACH, it consumes almost the same amount of power as CPU fully running at IDLE state. When the smartphone is in IDLE,

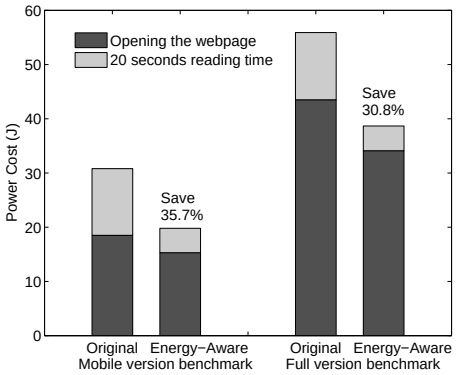


(a) Average

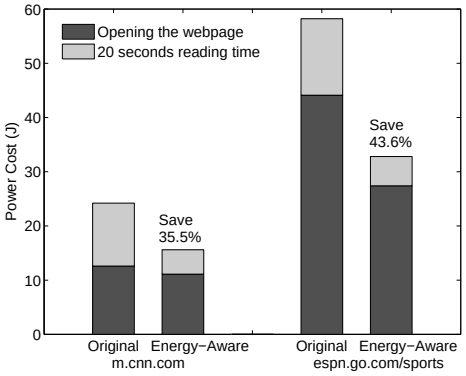


(b) Two popular webpages

Figure 6. The data transmission time



(a) Average



(b) Two popular webpages

Figure 7. The power consumption

Table IV
THE POWER CONSUMPTION OF THE SMARTPHONE AT DIFFERENT STATES (WHICH INCLUDES POWER FOR DISPLAY AND SYSTEM MAINTENANCE)

	Power (W)
IDLE state	0.15
FACH state	0.63
DCH state without transmission	1.15
DCH state with transmission	1.25
CPU fully running (IDLE state)	0.6

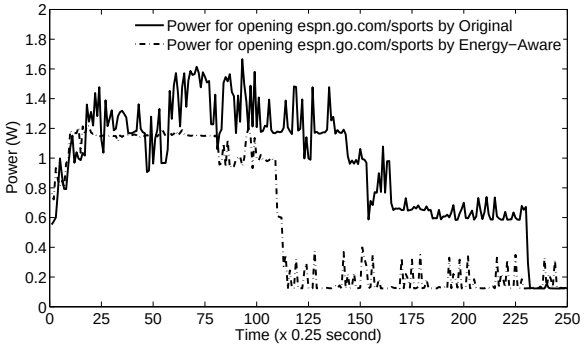


Figure 8. The power consumption for loading the ESPN sports webpage

the power is mainly consumed by screen display and system maintenance. Since our energy-aware approach reduces the time at FACH and DCH states, it can save power for web browsing.

Fig. 8 compares the power consumption of the original web browser and our energy-aware approach when loading the webpage espn.go.com/sports. The original web browser and our approach finish the data transmission at time 130 and 100, respectively. Our approach switches the smartphone from FACH state to IDLE state at time 100, and then the smartphone consumes very little power. On the other hand, the original web browser keeps the smartphone in FACH between time 150 and 230, which consumes 0.6W more power.

Suppose the reading time is longer than 20 seconds, in the original approach, the smartphone will stay in FACH for 20 seconds, but our approach can put the smartphone into IDLE. As shown in Fig. 7(a), with mobile version benchmark, our approach can reduce the power consumption by 35.7%. Most of this power saving comes from putting the smartphone into IDLE during the reading time. With full version benchmark, our approach can reduce the power consumption by 30.9%.

Different from the mobile version, most of this power saving comes from reducing the data transmission, which is achieved by separating the data transmission computation and layout computation.

We also evaluate the power saving with two web sites: the mobile version of m.cnn.com and the full version of espn.go.com/sports. As shown in Fig. 7(b), the general trend is similar to Fig. 7(a). Our approach reduces the power consumption by 35.5% for m.cnn.com, and reduces the power consumption by 43.6% for espn.go.com/sports.

browser, but the display in our approach is about 10.6 seconds

D. Predicting the reading time

In this subsection, we evaluate the performance of our prediction algorithm based on the collected data traces. We first measure the prediction accuracy and then measure the power saving achieved through predicting the reading time. Finally, we measure the computational cost of prediction.

1) Data Trace Collection: We distribute our prototype smartphones to 10 students and collect their web access history for one week. The web access history of each student forms a data trace, which includes at least 10 hours of web browsing activities. Each data trace records the webpage reading time and key features of web browsing. Webpages with reading time longer than 10 minutes are discarded, because we do not know whether the user is still reading the webpage after such a long time.

We measure the reading time in the data traces and show the cumulative distribution in Fig. 9. First, the reading time of 47% webpages is larger than the threshold T_p (9 seconds). For these webpages, switching the smartphone into IDLE can save power but incur some transmission delay. Second, the reading time of 32% webpages is larger than the threshold T_d (20 seconds). For these webpages, switching the smartphone into IDLE can save power without causing transmission delay. Third, the reading time of 30% webpages is smaller than 2 seconds. Hence, we set the interest threshold α to 2 seconds. With this interest threshold, we do not need to predict the reading time of these 30% webpages, because the user will access other webpages within the interest threshold.

Our prediction model is based on 10 features of webpages. We calculate the Pearson's correlation coefficient between the reading time and these 10 features and show the result in Table V. The correlation coefficients are quite small, which means that no notable correlation exists between the reading time and the 10 features. Besides, it also indicates that simple linear models are not suitable for predicting the reading time.

2) Prediction Accuracy: For each data trace, we choose the first half of the data trace to train our model, and use the other half to evaluate the prediction accuracy. Then we calculate the average over all data traces. We also

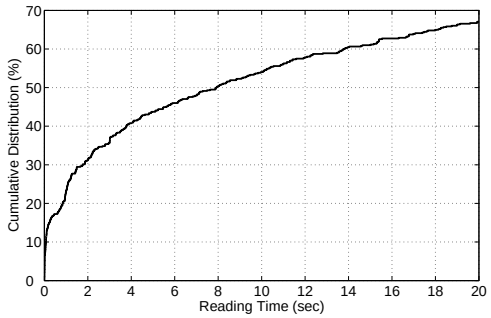


Figure 9. The cumulative distribution of the reading time of all the user data.

study prediction with and without considering the interest threshold. For prediction without considering the interest threshold, we train the prediction model with the original data trace. For prediction with interest threshold, we exclude the reading times that are shorter than 2 seconds.

If the predicted reading time and the corresponding real value are both larger or smaller than a given value (T_d or T_p), the prediction is correct. The *prediction accuracy* is defined as the percentage of correct predictions. The thresholds T_p and T_d are set to 9 and 20 seconds, respectively.

Fig. 11 shows the evaluation result of the average prediction accuracy. Using interest threshold can increase the average prediction accuracy by at least 10%.

3) Effects of prediction on energy and delay: Table VI shows 6 cases to represent 6 different conditions for switching the wireless interface to IDLE. We use the real reading time in the data traces (Accurate-9 and Accurate-20) as the predicted value to obtain the performance upper bound that our prediction-based approach can achieve. Since our energy-efficient approach can be power driven or delay driven, there are two thresholds: T_d = 9 seconds for the power driven approach and T_d = 20 seconds for the delay driven approach. We also consider the always-off approach for performance comparison. In the *Original Always-off* approach, the smartphone switches to IDLE immediately after the webpage is opened. The *Energy-Aware Always-off* is the proposed approach that reorganizes the computation sequence.

Fig. 12 shows the power saving and delay reduction of different approaches. The *Original Always-off* approach increases the delay by 3.3% instead of reducing the delay, because it increases the delay for all webpages whose reading time is less than 20 seconds. The *Energy-Aware Always-off* approach reduces the least amount of delay (by about 7.4%) among all other 6 approaches. The *Original Always-off* approach saves the least amount of power, because it can only save power during the reading time. Other approaches are built on our method, and thus can also save power during the webpage loading time. Among all approaches, *Accurate-20* reduces the most amount of delay (10%) and *Accurate-9* saves the most amount of power (23.5%). The power saving

Table V
PEARSON'S CORRELATION COEFFICIENT BETWEEN READING TIME AND 10 FEATURES

Transmission time	Page Size	Objects	Javascript Num	Figures Num	Figure Size	Javascript Time	Second URL	Height	Width
0.0009	0.059	0.023	0.042	0.013	0.015	0.021	0.038	0.067	0.016

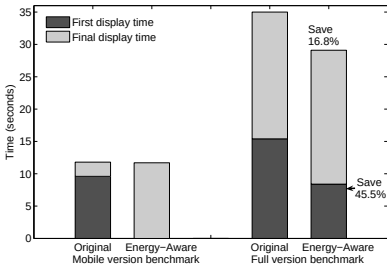


Figure 10. The average screen display time

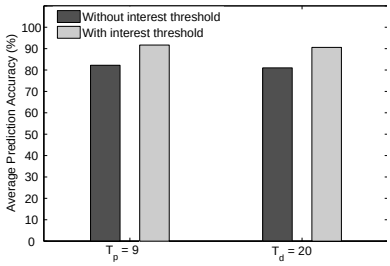


Figure 11. The average prediction accuracy with and without interest threshold

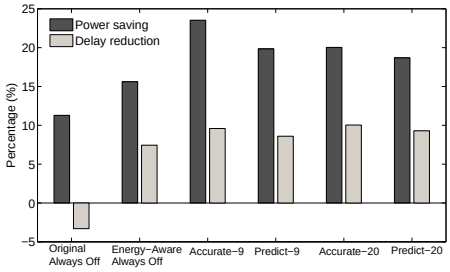


Figure 12. The power and delay saving of six cases

Table VI
THE CONDITION FOR SWITCHING TO IDLE

Case	Description
Original Always-off	After the webpage is opened by the original web browser
Energy-aware Always-off	After the webpage is opened in our approach where the computation sequence is reorganized
Accurate-9	The real reading time in the data trace is longer than $T_p = 9$ seconds in our approach
Predict-9	The predicted reading time is longer than $T_p = 9$ seconds in our approach
Accurate-20	The real reading time in the data trace is longer than $T_d = 20$ seconds in our approach
Predict-20	The predicted reading time is longer than $T_d = 20$ seconds in our approach

Table VII
POWER CONSUMPTION AND EXECUTION TIME OF THE PREDICTION ALGORITHM

Number of Decision Trees	1000	10000	20000
Power Consumption (J)	0.016	0.177	0.326
Execution Time (second)	0.027	0.295	0.543

and delay reduction achieved by *Predict-20* and *Predict-9* are quite close to those achieved by *Accurate-20* and *Accurate-9*. It means that our prediction algorithm is quite effective on power saving and delay reduction.

4) Computational Cost: We measure the power consumption and the execution time of different decision tree sizes used for prediction. Table VII shows that the prediction algorithm only takes 0.295 second and consumes 0.177 J power on average for going through 10000 decision trees with 8 nodes. Unlike the malware detection [10], we do not need to frequently train and upload the decision trees to the smartphone. Too frequently training the decision tree may lead to overfitting and degrade the generalization capability of the model.

VI. RELATED WORK

There have been lots of research on optimizing the power usage of smartphones [1], [2], but they only focus on reducing the power consumption of one component such as display. Although some of them also consider the power consumption of the wireless interface, most of them focus on the WiFi or Bluetooth interface [3], [11]–[13].

As the number of 3G smartphones exponentially increases, it has become an important problem to save the power of the 3G wireless interface. Bartendr [14] establishes the relationship between signal strength and power consumption, and schedules communication based on the signal strength to save power. The power consumption of mobile devices with different timer values has been measured by analytical models [15]. Instead of finding optimal values, some previous studies focus on changing timer values to save power in the FACH state. TOP [16] dynamically changes the timer value with input from the applications. Differently from these works, our approach updates the timer based on the predicted reading time. Although the reading time has been predicted and used for various information retrieval tasks [8], [17], none of them has applied to power saving in smartphones. Complementary to this work, Qian *et al.* [18] proposed caching techniques to reduce the power consumption and improve the radio resource utilization for web browsing in 3G networks. Zhao *et al.* [19] proposed an architecture called virtual-machine based Proxy (VMP), which shifts the computation from smartphones to VMPs to save power and delay for web browsing in 3G networks.

Grouping multiple transmissions in a burst has been used in [20], which groups transmissions by deferring them and hence increases the transmission delay. Many recent smartphones and networks support the fast dormancy feature included in 3GPP [21]. The smartphone can send a message to notify the network that its data transmission is complete, and thus the smartphone can switch to low power level.

However, fast dormancy requires the smartphone to notify the network. Furthermore, it cannot know when the next transmission will happen. As introduced in Section III-A, if the next transmission happens quickly, fast dormancy may waste power on switching the smartphone back to high power level.

Speculative parsing (used in Webkit-based browsers) [22] and Google SPDY [23] are techniques to reduce the webpage loading time. However, none of them considers reducing the power consumption of smartphones, and they are complement to our proposed technique. For example, in speculative parsing, while executing a script, another thread can parse the document and load another object to allow some parallelism. However, speculative parsing is limited due to dependency requirements in the DOM tree. This is different from our approach which passes the whole file to fetch all objects related to data transmission so that the wireless interface can enter sleep earlier.

Recently, how to reduce the webpage layout computation has received much attention, because it takes 40–70% of the processing time for loading webpages [5]. Meyerovich *et al.* [5] provides a fast layout engine to speed up the layout computation for new webpages. For webpages that have already been opened, Zhang *et al.* [24] propose a layout caching approach. It caches the layout results to eliminate redundant computations and achieves more efficient local webpage processing. Different from them, our approach focuses on reorganizing the computation sequence to reduce the total data transmission time to save power.

VII. Conclusions

In this paper, we proposed an energy-aware approach for web browsing in 3G based smartphones. First, we reorganize the computation sequence for loading webpage so that the web browser can first run the computations that will generate new data transmissions and retrieve these data. Then, the web browser can put the 3G radio interface into IDLE, release the radio resource, and then run the remaining layout computation. This not only saves power, but also reduces the webpage loading time by removing the computation intensive redraws and reflows. Second, we predict the user reading time on the webpage after it is downloaded. If this time is longer than a threshold, the radio interface can be put into IDLE to save power. Since smartphones have limited computation capability, we propose a low overhead prediction algorithm based on Gradient Boosted Regression Trees (GBRT). Experimental results show that our approach can reduce the power consumption of smartphone by more than 30% during web browsing, and reduce the webpage loading time by 17%.

References

[1] J. Sorber, N. Banerjee, M. D. Corner, and S. Rollins, "Turducken: hierarchical power management for mobile devices," *Int'l Conf. on Mobile Systems, Applications, and Services (MobiSys)*, 2005.

[2] M. Dong and L. Zhong, "Chameleon: a color-adaptive web browser for mobile OLED displays," MobiSys, 2011.

[3] J. Liu and L. Zhong, "Micro power management of active 802.11 interfaces," in *MobiSys*, 2008.

[4] *3GPP Specification TS 24.008: Mobile radio interface Layer 3 specification; Core network protocols; Stage 3*, Std., Rev. Rel. 9. [Online]. Available: http://www.3gpp.org.

[5] L. A. Meyerovich and R. Bodik, "Fast and parallel webpage layout," in *Int'l Conf. on World wide web (WWW)*, 2010.

[6] J. H. Friedman, "Greedy function approximation: A gradient boosting machine cite," in *Annals of Statistics*, 1999, pp. 579–588.

[7] J. P. Romero, O. Sallent, R. Agusti, and M. A. Diaz-Guerra, *Radio resource management strategies in UMTS*. John Wiley and Sons, Inc, 2005.

[8] C. Liu, R. W. White, and S. Dumais, "Understanding web browsing behaviors through weibull analysis of dwell time," *ACM SIGIR*, 2010.

[9] A. I. Inc. (2009) The top 500 sites on the web. Http://www.alexa.com/topsites.

[10] A. Bose, X. Hu, K. G. Shin, and T. Park, "Behavioral detection of malware on mobile handsets," *MobiSys*, 2008.

[11] F. R. Dogar, P. Steenkiste, and K. Papagiannaki, "Catnap: exploiting high bandwidth wireless interfaces to save energy for mobile devices," *MobiSys*, 2010.

[12] H. Zhu and G. Cao, "On Supporting Power-Efficient Streaming Applications in Wireless Environments," *IEEE Transactions on Mobile Computing*, July/August 2005.

[13] W. Hu, G. Cao, S. Krishanamurthy, and P. Mohapatra, "Mobility-Assisted Energy-Aware User Contact Detection in Mobile Social Networks," *IEEE Int'l Conf. on Distributed Computing Systems (ICDCS)*, 2013.

[14] A. Schulman, V. Navda, R. Ramjee, N. Spring, P. Deshpande, C. Grunewald, K. Jain, and V. N. Padmanabhan, "Bartendr: a practical approach to energy-aware cellular data scheduling," *ACM MobiCom*, 2010.

[15] J.-H. Yeh, J.-C. Chen, and C.-C. Lee, "Comparative analysis of energy saving techniques in 3GPP and 3GPP2 systems," *IEEE transactions on vehicular technology*, 2009.

[16] F. Qian, Z. Wang, A. Gerber, Z. M. Mao, S. Sen, and O. Spatscheck, "Top: Tail optimization protocol for cellular radio resource allocation," *IEEE ICNP*, 2010.

[17] J. Attenberg, S. Pandey, and T. Suel, "Modeling and predicting user behavior in sponsored search," *ACM SIGKDD Int'l Conf. on Knowledge discovery and data mining*, 2009.

[18] F. Qian, K. S. Quah, J. Huang, J. Erman, A. Gerber, Z. Mao, S. Sen, and O. Spatscheck, "Web caching on smartphones: ideal vs. reality," *MobiSys*, 2012.

[19] B. Zhao, B. C. Tak, and G. Cao, "Reducing the delay and power consumption of web browsing on smartphones in 3G networks," *IEEE ICDCS*, 2011.

[20] N. Balasubramanian, A. Balasubramanian, and A. Venkataramani, "Energy consumption in mobile phones: a measurement study and implications for network applications," *ACM Internet Measurement Conference (IMC)* 2009.

[21] *Configuration of fast dormancy*, Std., Rev. Rel. 8. [Online]. Available: http://www.3gpp.org

[22] (2011) Webkit's speculative parsing. Http://gent.ilcore.com/2011/01/webkit-preloadscanner.html.

[23] (2012) Google spdy. Http://en.wikipedia.org/wiki/SPDY.

[24] K. Zhang, L. Wang, A. Pan, and B. B. Zhu, "Smart caching for web browsers," *Int'l Conf. on World wide web (WWW)*, 2010.