Network Delay-Aware Energy Management for Mobile Systems

Minho Ju*†, Hyeonggyu Kim*, and Soontae Kim*
*School of Computing, KAIST, Daejeon, Korea
†Samsung Electronics Co., Ltd., Suwon, Korea
*{minhoju, hyeonggyu, kims}@kaist.ac.kr, †minho7.ju@samsung.com

***Abstract*—Smartphones and tablets have occupied the every facet of our daily life in recent years. According to a recent survey, users spend over 3 hours a day on their mobile devices. In addition, 76% and 75% of smartphone users perform web browsing and social networking at least once a day, respectively. To fully enjoy their benefits, those mobile systems require a long battery life. However, network errors such as packet losses decrease the battery life more quickly. We analyzed the reason for this through measurements using real smartphones and mobile full system simulation. We found that the smartphones maintain high performance level on packet losses without doing useful work. To address this problem, we propose a method for reducing energy consumption by lowering down performance level with a Dynamic Voltage and Frequency Scaling mechanism when long network delay is expected due to packet losses. Experimental results show that the total energy consumption is reduced by 8.4% without performance loss.**

I. Introduction

Mobile systems such as smartphones and tablet PCs have come into wide use because they can aid in the following aspects of everyday life using a wireless Internet connection: human relationships, work, shopping, searching for information, and providing news. According to a recent report related to mobile user behavior [1], mobile users spend 3.3 hours a day on average on their smart devices, and 85% of users said that mobile systems are a central part of their every life. Smartphones are used at least once a day for searching the Internet and social networking by 76% and 75% of smartphone users, respectively. Total mobile activity including mobile browser usage recently eclipsed 60% of all computer use, with desktop computers accounting for the remaining 40%. Moreover, the network traffic using mobile systems has been rapidly increasing owing to the presence of social networking services such as Facebook, Tumblr, Pinterest, Instagram, and Vince, which combined simple texts with images and video clips. An *et al.* [2] at DOCOMO Euro-Labs studied mobile Internet usage behavior and found that the ratios of HTTP requests related to the web and SNSs were 69% and 12%, respectively. Accordingly, analyzing the behavior of network-based applications such as mobile web browsers is very important.

The web loading time of mobile web browsers has increased because the resources used by web pages have increased in number and complexity [3]. In mobile web sites, 64.5% of web resources are images in terms of both their sizes used and the number of objects [3]. Furthermore, losses or errors of network packets occur in the real world, consequently increasing the loading time owing to the need for retransmissions. In Google service servers [4], 6.1% of HTTP replies are lost, and 10% of all TCP connections include at least one loss. The average server retransmission rate is 2.5%. Such losses make the web access latency period five times longer, which most importantly increase energy consumption. Modern mobile systems such as Android smartphones do not consider the energy consumption of hardware components, except the network components while handling network errors. For example, when network errors occur during web browsing, the systems show only a progress bar for the loading of web pages until a predefined time, while the screen is still turned on. The systems simply wait for retransmitted packets, thereby consuming unnecessary energy. The optimization of energy consumption is very important in mobile systems because of their limited battery capacity. The battery life together with overheating still tops the list of complaints charts for smartphone users tired of their daily charging routine [5].

To address the above limitations, we propose a network delay-aware scheme with a *duplicate ACK handler* and a new DVFS governor called *OnNetwork*. The increased latency by network errors is fairly long in the perspective of of the processing speed of computer components such as CPU and caches. To suppress the extensive and unnecessary operations of applications such as updates of the graphical user interface and intermediate web pages that have yet to finish loading, our scheme maintains low CPU frequency and voltage on occurrences of network errors. The followings are our key contributions. First, we characterize an energy consumption and performance depending on network packet loss rates based on the measurements using a real device and architectural simulation. Second, we propose a method for detecting network delay by errors in the Linux Kernel. Third, we propose a method for reducing the energy consumption using a network delay-aware DVFS governor. Finally, we show experiments that conducted through loading of ten websites demonstrate that the proposed scheme reduces the energy consumption by average 8.4% for a real packet loss rate of 1.18%.

The rest of this paper is organized as follows. Section II introduces the previous work and Section III analyzes the problems in handling network errors in the modern mobile systems. In Section IV, we describe our proposed scheme consisting of a *duplicate ACK handler* and *OnNetwork governor*, which are implemented in the Linux Kernel. Section V discusses the experimental results, and concluding remarks are given in Section VI.

978-3-9815370-7-9/DATE16/

TABLE I. OVERVIEW OF ANALYSIS TOOL AND SIMULATION.

Category	Description	Tools
Architectural analysis	Architectural full-system simulator	gem5
	Power modeling	gem5, McPAT
Observation on a real device	Measure the power for Nexus 5	SmartPower
	Capture network packets	tcpdump
	Analyze network packets	Wireshark
	Trace graphical processing events	DDMS
	Trace Kernel events (scheduling, power)	Ftrace
Network condition modeling	Create network environments in which errors occur	netem
Benchmark	Tests browser page loading performance	BBench

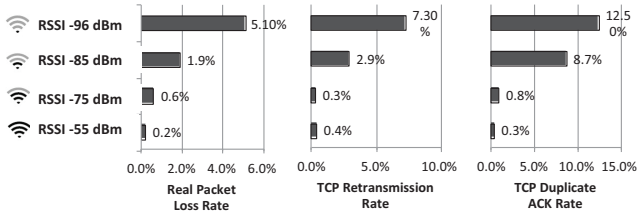


Fig. 1. Network packet loss rates (left), TCP retransmission rates (middle), and TCP duplicate ACK rates (right) of the captured network packets depending on the WiFi signal strength in a Nexus 5 device while loading `BBench` [6] websites from our web server. We emulated the packet loss rates using the *netem* tool.

II. RELATED WORK

Power analysis of network errors. The authors of [7], [8], [9] studied the characteristics and impacts on the energy consumption and performance of the wireless networks based on real measurements. Ding *et al.* [7] showed the relationship between the battery drain and wireless signal strength. They revealed that the 3,785 smartphones used in their experiments transferred their data when 3G and WiFi signal strengths are poor for 43% and 21% of time, respectively. Huang *et al.* [8] indicated that the performance bottleneck of web-based applications is not only the network but also the processing power of the devices. Huang *et al.* [9] showed the network effects on web browsing. However, they only showed the total energy consumption of the mobile system and excluded the network module. We analyze the causes of these problems for each application using a mobile full system simulator.

Reducing CPU, DRAM, and network energy consumption. To reduce energy consumption in on-chip caches and DRAM, which are major energy consumers in CPUs, a small L0 data cache and DRAM traffic clustering schemes are proposed in [10], [11]. Our proposed scheme can also reduce their energy consumption by reducing unnecessary activities in them. The technique in [12] controls the CPU frequency and wireless interface selection between WiFi and 3G to reduce energy consumption depending on the network bottleneck and application scheduling. During their simulation, the authors considered a network bottleneck using only throughput estimation. Our proposed approach differs from this technique in detecting network delay, as discussed in Section III.

Reducing network power by offloading. Zhao *et al.* [13] proposed a method called a virtual-machine-based proxy (VMP) that shifts computation from mobile systems to VMPs to reduce the energy consumption and delay for web browsing in a 3G network. However, they covered only the network delay in 3G networks. Our scheme can also cover delay by network errors.

Energy-efficient web browsing. In [14], [15], the authors proposed a web browsing method for high performance and energy efficiency. By predicting the reading time of a web page, the scheme in [14] utilizes the network states with different powers to reduce the web page loading time and increase the network capacity. For big and little CPUs, the scheme in [15] identifies and schedules web pages using an ideal core and frequency configuration for energy consumption reduction. However, these works are limited to web browsing. Our scheme can be applied to all applications using networks, because our scheme considers the network errors and controls the CPU frequency in the Linux Kernel.

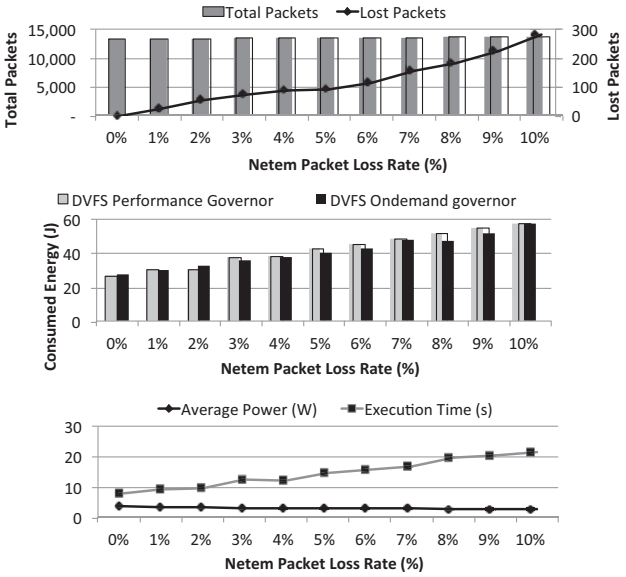


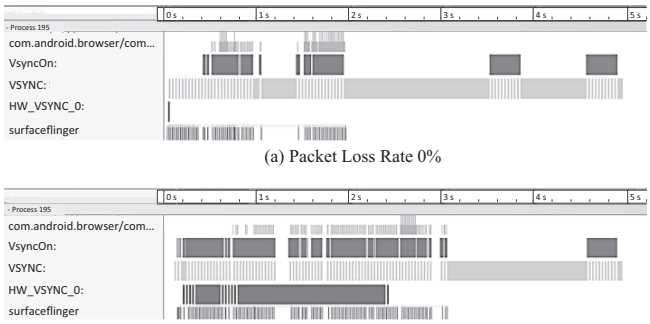
Fig. 2. The numbers of captured and lost packets (upper), energy consumption (middle), and average power and execution time (bottom) depending on the packet loss rates as the input value of the *netem* tool. The real packet loss rates for netem packet loss rates from 1% to 10% are 0.20%, 0.41%, 0.57%, 0.66%, 0.71%, 0.86%, 1.18%, 1.37%, 1.70%, and 2.10%, respectively. For lost packets, look at the right y axis in the upper figure.

III. BACKGROUND AND MOTIVATION

To analyze how a mobile system such as the Android platform behaves on network delay or error occurrences, we used analysis tools and an architectural mobile full system simulator [16], as shown in TABLE I. We target mobile web browsing because this is a common user activity when using mobile Internet. To show energy consumption reduction, we used `BBench` [6], which is an automated benchmark that tests browser's page rendering performance. To analyze the impact of network delay, we modified the BBench to measure only the time required for loading web pages. To create a network environment in which errors occur similar to the real world, such as a packet loss and corruption, we used the `netem` tool, which operates in the Linux Kernel within our web server.

A. Handling Network Delay in Mobile Systems

Today, the transmission control protocol (TCP) is the most commonly used Internet protocol. TCP guarantees reliable communications for applications such as browsers, Facebook, and messenger services. TCP handles network errors such as packet losses and corruptions using sequence numbers and acknowledgments (ACKs). In particular, because a mobile



(a) Packet Loss Rate 0%

(b) Packet Loss Rate 10%

Fig. 3. Graphical events trace for screen updates using the *DDMS* tool. The bars represent execution of each process. Note that when packet loss rate is 10%, more graphical events occur, which consume energy unnecessarily.

system has the role of a client, a mobile system sends a duplicate ACK for fast retransmission if some packet gets lost. As the packet loss rate increases, TCP retransmission rate and duplicate ACK rate increase, as shown in Fig. 1, in which we captured network packets using the *tcpdump* tool and analyzed them using the *Wireshark* tool.

These packet losses generate long web access latencies. A flow with a packet loss takes on average five times longer latency to complete than that without such a loss [4]. This is because the server side retransmits the lost packets using a retransmission timer based on RTTs (round trip times) for a packet loss recovery. Using a measurement of billions of TCP connections from clients to Google services, the authors in [4] found that nearly 10% of them incur at least one packet loss. Furthermore, 77% of the losses are repaired through expensive retransmission timeouts (RTOs), often because packets at the tails of a burst are lost, preventing a fast recovery. Consequently, packet loss recovery dominates the web latency, thereby increasing energy consumption.

B. Energy Consumption depending on Packet Loss Rates

We measured the energy consumption of a Nexus 5, which maintains the darkest screen brightness, using an `ODROID Smart Power` device depending on packet loss rates, as shown in Fig. 2. We confirmed that 13,313 packets were captured from ten websites: Amazon, BBC, CNN, Craigslist, Ebay, Google, MSN, Slashdot, Twitter, and YouTube. For analysis depending on packet loss rates, we configured the loss rates of the netem tool from 0% to 10% by increasing 1% each step. However, the captured loss rates using the tcpdump tool are lower than the packet loss rate of the netem tool, as shown in Fig. 2. For example, the captured loss rate corresponding to the 7% netem loss rate is 1.18%.

We can see that even for only a 1% netem packet loss rate, the energy consumption increases by 8.6% compared with no packet loss. In the case of 10% netem packet loss rate, the percentage of increased energy consumption is 109.1%. Moreover, the energy consumption of the OnDemand governor as the default DVFS governor is similar to that of the DVFS performance governor, which maintains the highest CPU frequency. As the packet loss rate increases, the average power consumption is similar, whereas the execution time increases in proportion to the packet loss rate. **We can see that while the network errors occur, the system does not strive to save the energy consumption.**

TABLE II. EXECUTION TIMES OF THE BROWSER, GRAPHICS, AND WIFI THREADS WHILE LOADING BBENCH ON A NEXUS 5.

Thread	Packet Loss 0%	Packet Loss 10%	Increased Rate
Chrome_IOThread	219.2 ms	273.0 ms	24.6 %
Chrome_ChildIOT	40.4 ms	47.5 ms	17.6 %
SimpleCacheWork	62.3 ms	75.3 ms	21.2 %
SurfaceFlinger	113.0 ms	179.4 ms	58.8 %
EventThread	21.3 ms	27.0 ms	26.4 %
DispSync	13.4 ms	18.8 ms	40.0 %
dhd_dpc (WiFi)	98.5 ms	150.2 ms	52.6 %
dhd_rxf (WiFi)	45.3 ms	55.9 ms	23.5 %

C. Analysis using Real Measurements and Mobile Full System Simulator

To know which application processes' execution times are increased during network errors, we traced the execution time of each process using the *FTrace* tool, as shown in TABLE II, on a Nexus 5. We can see that the execution times of the processes related to graphics processing such as surfaceflinger are considerably increased. In addition to sufaceflinger, the executions of processes related to VSync are increased, which generates periodically events at fixed interval for handling user inputs, animations, and UI updates, as shown in Fig. 3. We can also confirm that the surfaceflinger process, which performs screen update events, executed quietly on network errors. Moreover, the executions of the web browser are also increased for updating a progress bar.

For a more accurate and precise analysis, we conducted a mobile full system simulation while operating the Android system with the Linux Kernel as configured in TABLE III, which is similar to a Nexus 5 device. We constructed a system configuration in which an Android system and a web server are connected during simulation using a simulated network with the gem5 simulator. For the validation of the mobile full system simulator, we measured the load time of BBench web pages. As a result, we confirmed that the absolute average error rate is 26.8%, as shown in Fig. 4. Because of dynamic real network condition and inaccuracy of server modeling, the error rate is a bit high but low enough to observe what occurs on packet losses.

Fig. 5 shows the characterization of each process execution

TABLE III. MOBILE FULL SYSTEM SIMULATION CONFIGURATION.

Execution core	300 MHz ∼ 2.26 GHz 4-core CMP, ARM ISAs, out of order, tournament branch predictor, 4,096 BTB entries, 64 reorfer buffer, 32 fetch queue
Caches	L1 I-cache: 16 KB/4 way, private, 64B block size, LRU, 4 MSHRs L1 D-cache: 16 KB/4 way, private, 64B block size, LRU, 4 MSHRs L2 cache: 2 MB/8 way, shared, 64 bytes block size, random, 20 mshrs
DRAM	LPDDR3_1600x32, 2 GB, 2 channels, 32 bus width, 8 banks per rank, 4 KB row buffer size, 1 rank per channel
TLB	ITLB: ArmTLB type, 32 entry DTLB: ArmTLB type, 32 entry
Display	1920 x 1080 resolution
HPM	28nm technology
Android	Jelly Bean 4.1
Kernel	Linux 3.14

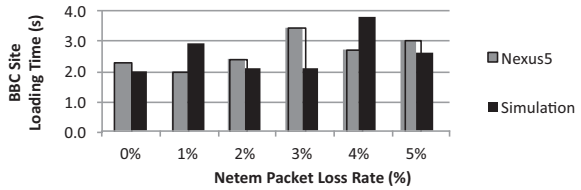
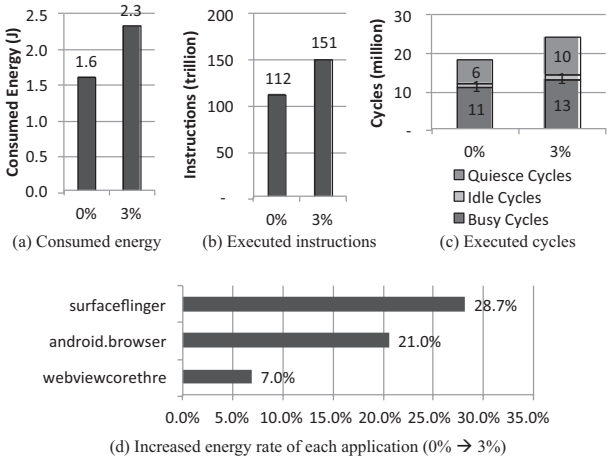


Fig. 4. Web page loading times of BBC website on a Nexus 5 device and in a mobile full system simulation for gem5 simulator validation.



(a) Consumed energy (b) Executed instructions (c) Executed cycles

(d) Increased energy rate of each application (0% → 3%)

Fig. 5. Comparisons of consumed energy, executed instructions, cycles, and process energy consumptions based on the mobile full system simulation when the packet loss rates are 0% and 3% as the input of the *netem* tool. Note that the surfaceflinger and android.browser for updating the progress bar and intermediate web rendering images incur many operations, thereby consuming more energy consumption. Quiesce cycles are the total number of cycles the CPU spends de-scheduled due to a pause operation or waiting for an interrupt.

and the architectural behaviors when the packet loss rate is increased to 3% compared to when there is no packet loss. The overall percentages of increased energy consumption and executed instructions are 43.8% and 34.8%, respectively. We can confirm that the system is not in an idle state during network errors, because the number of CPU cycles is also increased by 33.3%. In addition, the surfaceflinger, android.browser, and webviewcorethread processes, which are responsible for graphical updates and web page rendering, consume more energy consumption. These processes execute many instructions and function calls, and thus they make cache and TLB accesses, which consume dynamic energy. Our analysis revealed that the progress bar generates extensive update executions, and thus many instructions. Moreover, webviewcorethread as a render creates and shows intermediate web pages of unfinished rendering images. Such processing makes the DVFS OnDemand governor to maintain the highest CPU frequency.

IV. Proposed Scheme

A. Design Methodology

The key approach for reducing energy consumption is to limit the CPU operations to be performed when a network delay occurs. To this end, we modified the TCP protocol in the network module and added a new DVFS governor into the cpufreq module in the Linux Kernel. Our proposed scheme consists of two parts, as shown in Fig. 6. First, to notify the system of occurrences of network errors, we modified the TCP kernel codes to notify the receivers of the expected delay. Second, to reduce energy consumption, we made a new DVFS governor called *OnNetwork*. The OnNetwork governor normally operates like the *OnDemand* governor, but it maintains the minimum frequency during a network delay. In addition, we designed a duplicate ACK handler as a framework to send events to the registered modules called duplicate ACK receivers in the Linux Kernel.

The method used to detect a network delay is described

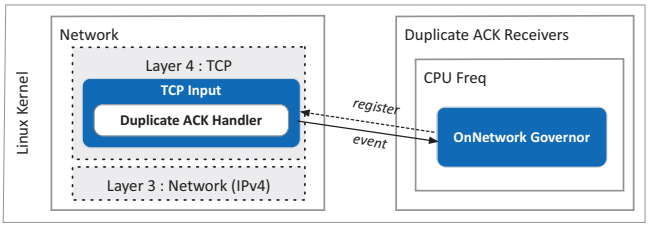


Fig. 6. Our proposed scheme in the Linux Kernel.

in Algorithm 1. TCP duplicate ACK handler processes two events, the Round-Trip Time (RTT) measurements and network errors, when TCP duplicate ACK receivers such as the OnNetwork governor are registered. If a measured RTT event occurs, the handler calculates a network delay. We use the smoothed round-trip time [17] by averaging sample packets as the expected wait time instead of the medium or maximal deviation of the round-trip time. If a network error event occurs, it notifies the network delay to a duplicate ACK receiver. Because mobile systems are usually clients in a network connection, we use a duplicate ACK caused by a lost or corrupt packet and reordering of packets. The TCP protocol in the server considers that a packet has been lost, if three or more duplicate ACKs are continuously received.

Algorithm 2 shows a method for controlling the CPU DVFS during a network delay. When the OnNetwork governor receives a network delay event from the TCP duplicate ACK handler, it calculates the expected wait time. We designed a parameter ratio r, which is the time to control CPU frequency for reducing energy consumption and preventing performance degradation. Our OnNetwork governor is based on the OnDemand governor, which is the default governor of a Nexus 5. Thus, if no network delay occurs, the OnNetwork governor operates the same as the OnDemand governor. However, if the current time is within the expected wait time, it sets the CPU frequency to the minimum value, which is variable according to CPU policy by the system environment such as CPU temperature.

We did not change the original TCP mechanism in terms of either portability or compatibility. Moreover, our scheme does not impact the user experiences. Because our algorithms are implemented in the Linux Kernel, our scheme can be applied to systems other than Android that use this Kernel, such as wearable computing and IoT.

B. Implementation Details in the Linux Kernel

We modified two files of the original Kernel code, `tcp.h` and `tcp_input.c`, and added a file, `cpufreq_onnetwork.c`, as a Kernel module. To detect a network delay, as described in Algorithm 1, we modified the `tcp_rtt_estimator` function, which calculates an estimated RTT and its medium deviation, to know the delay when a packet is lost. We can detect the time required to send a duplicate ACK in the functions `tcp_dsack_set` and `tcp_sack_extend`. We modified theses two functions for notifying the occurrence times of network errors along with the expected delay time required for the duplicate ACK receivers such as our DVFS OnNetwork governor.

For controlling the CPU frequency during a network delay, as described in Algorithm 2, we added our OnNetwork gover-

Algorithm 1 Pseudo code for *TCP Duplicate ACK Handler*

Require: A event e, which is a RTT measurement or a network error such as a packet loss and a corruption.
Let $L_{receivers}$ be the list of duplicate ACK receivers.
Let T_{rtt} be the delay time of the average round-trip.
repeat
  if e is the rtt measurements **then**
    for all the measured RTTs $mrtt$ **do**
      Calculate a smoothed round-trip time T_{smooth}.
      $T_{rtt} \leftarrow T_{smooth}$
    end for
  else if e is the network error **then**
    for all the duplicate ACKs **do**
      Reset T_{error} to the current time.
      Notify T_{error} and T_{rtt} to $L_{receivers}$.
    end for
  end if
until No registered duplicate ACK receivers.

Algorithm 2 Pseudo code for DVFS *OnNetwork* governor

Require: Registered as a duplicate ACK receiver when the system inits this Kernel module.
Let T_{wait} be the expected wait time due to network delay.
Let r be the time ratio of the received RTT (50%).
Let f be the CPU frequency during network delay.
procedure ONNETWORKDELAYEVENT(T_{error}, T_{rtt})
  $T_{wait} \leftarrow T_{error}$ + ($T_{rtt} \times r$)
end procedure
procedure DBSCHECKCPUONNETWORK($cpuPolicy$)
  repeat
    Reset $T_{current}$ to the current time.
    if $T_{current} < T_{wait}$ **then**
      $f \leftarrow$ the minimum frequency of $cpuPolicy$
      SETCPUFREQ($cpuPolicy$, f)
    else
      DBSCHECKCPU($cpuPolicy$) ▷ operate as *OnDemand* governor
    end if
  until DVFS timer exit.
end procedure

nor into the cpufreq module. When the OnNetwork governor receives the occurrence time of network errors from the TCP duplicate ACK handler in the `on_network_delay_event` function, it keeps the CPU frequency at the minimum value of CPU policy in the `dbs_check_cpu_onnetwork`. If CPU is not in an idle state, the `dbs_check_cpu_onnetwork` function is called periodically by the timer. This function checks the current time and the expected delay time at the front. If there is no network delay, this function call the `dbs_check_cpu` function, which operates same as the OnDemand governor.

V. EXPERIMENTAL RESULTS

To evaluate our proposed scheme, we measured the energy consumption of a Nexus 5 running Android 4.4 KitKat and Linux Kernel 3.4.0 with an `ODROID Smart Power` device at a 100-ms sampling rate. For accuracy, we measured the power of the system at the darkest brightness level of the display. We used the ten web sites in BBench described in Section III.

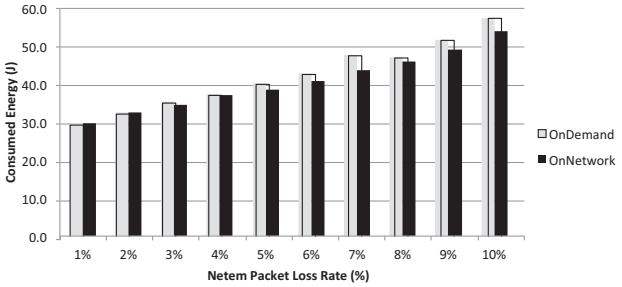


Fig. 7. The results of the total energy consumed when applying our DVFS *OnNetwork* governor with a *Duplicate ACK Handler* scheme as compared with the *OnDemand* governor when loading ten `BBench` websites.

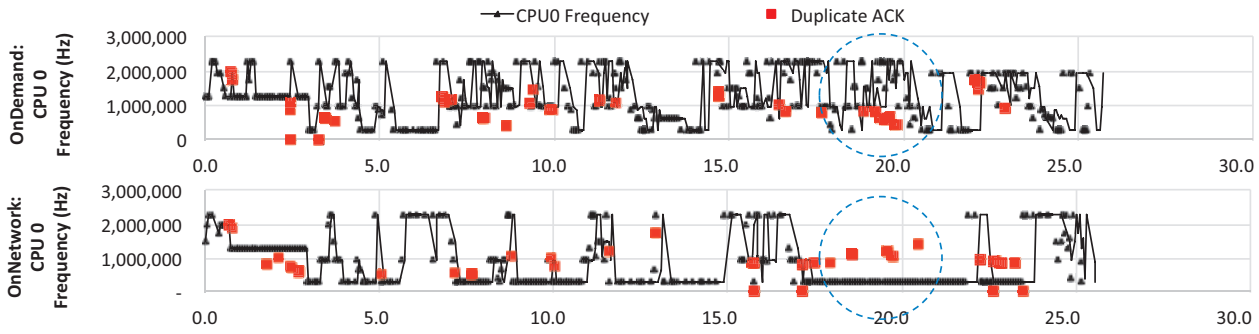
Impact of CPU frequency. Fig. 8 plots the frequency behavior example of our OnNetwork governor in contrast to the OnDemand governor when ten web sites are loaded under a packet loss rate of 10%. The square dots represent the times when the system sends a duplicate ACK. We can see that during the packet retransmission by the duplicate ACKs, the OnDemand governor does not drop the CPU frequency because the instructions for graphical tasks make the CPU busy, thereby incurring unnecessary energy consumption, whereas the OnNetwork governor drops the CPU frequency during the network errors to reduce unnecessary energy consumption.

Reduced energy. Fig. 7 shows the energy consumption savings of the proposed approach for each packet loss rate of the netem tool. In the case of the netem packet loss rate of 7%, which is 1.18% in reality, the overall energy consumption of the device is reduced by 8.44% on average, whereas there is no change in the execution time.

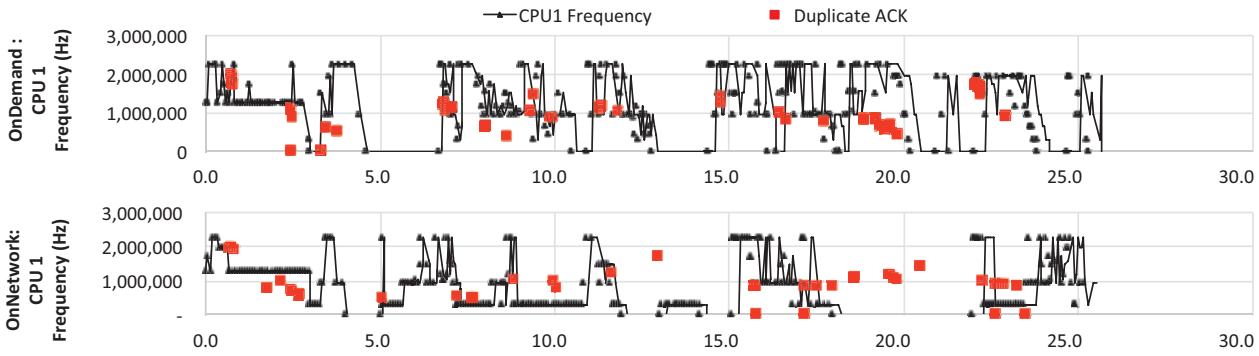
Impact of a design parameter. From our experiments, we determined the expected delay time, which is a parameter ratio *r* in Algorithm 2, for network errors to be one-half of the smoothed round-trip time from our experiments. Because network errors generally occur continuously, we found that the CPU frequency remains at a minimum without frequent switches, although the expected wait time is shorter than the real delay time. This parameter setting can prevent a degraded performance because the CPU can prepare processing just before receiving a network packet.

VI. CONCLUSION

In this paper, we analyzed the problems in handing a network delay owing to the occurrences of network errors, which consume unnecessary energy, using measurements on a real device and a mobile full system simulator. We found that extensive graphical updates and intermediate web page rendering are executed, and thus CPU remains at the high frequency. We proposed an energy-efficient handing method to be applied during network errors based on our *duplicate ACK handler* and new DVFS *OnNetwork* governor. Our scheme prevents the excessive tasks such as the processing of graphics and unnecessary processing, which keep the CPU frequency high while waiting for the packet retransmissions. The total energy consumption is reduced by 8.44% on average at a real packet loss rate of 1.18%.



(a) CPU0 frequency comparison between OnDemand (upper) and OnNetwork (bottom) governor



(b) CPU1 frequency comparison between OnDemand (upper) and OnNetwork (bottom) governor

Fig. 8. A trace example of controlling the CPU0 and CPU1 frequencies with the DVFS *OnDemand* and *OnNetwork* governors when loading `BBench` websites for the first 10 s with an netem packet loss of 10%. The zero frequency value means that its CPU state is idle. Dotted circles indicate the time period in which the difference between OnDemand and OnNetwork governors is clear.

ACKNOWLEDGMENT

This work was supported by ICT R&D program of MSIP/IITP [10041313, UX-oriented Mobile SW Platform] and the National Research Foundation (NRF) grant funded by Korean Government (No. 2014R1A2A2A01007051).

REFERENCES

[1] Salesforce, 2014 Mobile Behavior Report. [Online]. Available: http://www.exacttarget.com/sites/exacttarget/files/deliverables/etmc-2014mobilebehaviorreport.pdf [Accessed: August 7, 2015].

[2] X. An and G. Kunzmann, "Understanding mobile internet usage behavior," in *Networking Conference, 2014 IFIP*. IEEE, 2014, pp. 1–9.

[3] F. Qian, S. Sen, and O. Spatscheck, "Characterizing resource usage for mobile web browsing," in *Proceedings of the 12th annual international conference on Mobile systems, applications, and services*. ACM, 2014, pp. 218–231.

[4] T. Flach, N. Dukkipati, A. Terzis, B. Raghavan, N. Cardwell, Y. Cheng, A. Jain, S. Hao, E. Katz-Bassett, and R. Govindan, "Reducing web latency: the virtue of gentle aggression," in *ACM SIGCOMM Computer Communication Review*, vol. 43, no. 4. ACM, 2013, pp. 159–170.

[5] Fixya, Black Friday Smartphone Report. [Online]. Available: http://www.fixya.com/reports/blackfriday [Accessed: September 4, 2015].

[6] A. Gutierrez, R. G. Dreslinski, T. F. Wenisch, T. Mudge, A. Saidi, C. Emmons, and N. Paver, "Full-system analysis and characterization of interactive smartphone applications," in *Workload Characterization (IISWC), 2011 IEEE International Symposium on*. IEEE, 2011, pp. 81–90.

[7] N. Ding, D. Wagner, X. Chen, A. Pathak, Y. C. Hu, and A. Rice, "Characterizing and modeling the impact of wireless signal strength on smartphone battery drain," in *ACM SIGMETRICS Performance Evaluation Review*, vol. 41, no. 1. ACM, 2013, pp. 29–40.

[8] J. Huang, F. Qian, A. Gerber, Z. M. Mao, S. Sen, and O. Spatscheck, "A close examination of performance and power characteristics of 4g lte networks," in *Proceedings of the 10th international conference on Mobile systems, applications, and services*. ACM, 2012, pp. 225–238.

[9] J. Huang, Q. Xu, B. Tiwana, Z. M. Mao, M. Zhang, and P. Bahl, "Anatomizing application performance differences on smartphones," in *Proceedings of the 8th international conference on Mobile systems, applications, and services*. ACM, 2010, pp. 165–178.

[10] J. Lee and S. Kim, "An energy-delay efficient 2-level data cache architecture for embedded system," in *Proceedings of the 2009 ACM/IEEE international symposium on Low power electronics and design*. ACM, 2009, pp. 343–346.

[11] Y. Lee and S. Kim, "Dram energy reduction by prefetching-based memory traffic clustering," in *Proceedings of the 21st edition of the great lakes symposium on Great lakes symposium on VLSI*. ACM, 2011, pp. 103–108.

[12] J. Kwak, O. Choi, S. Chong, and P. Mohapatra, "Dynamic speed scaling for energy minimization in delay-tolerant smartphone applications," in *INFOCOM, 2014 Proceedings IEEE*. IEEE, 2014, pp. 2292–2300.

[13] B. Zhao, B. C. Tak, and G. Cao, "Reducing the delay and power consumption of web browsing on smartphones in 3g networks," in *Distributed Computing Systems (ICDCS), 2011 31st International Conference on*. IEEE, 2011, pp. 413–422.

[14] B. Zhao, W. Hu, Q. Zheng, and G. Cao, "Energy-aware web browsing on smartphones," *Parallel and Distributed Systems, IEEE Transactions on*, vol. 26, no. 3, pp. 761–774, 2015.

[15] Y. Zhu and V. J. Reddi, "High-performance and energy-efficient mobile web browsing on big/little systems," in *High Performance Computer Architecture (HPCA2013), 2013 IEEE 19th International Symposium on*. IEEE, 2013, pp. 13–24.

[16] A. Gutierrez, J. Pusdesris, R. G. Dreslinski, T. Mudge, C. Sudanthi, C. D. Emmons, M. Hayenga, and N. Paver, "Sources of error in full-system simulation," in *Performance Analysis of Systems and Software (ISPASS), 2014 IEEE International Symposium on*. IEEE, 2014, pp. 13–22.

[17] P. Karn and C. Partridge, "Improving round-trip time estimates in reliable transport protocols," in *ACM SIGCOMM Computer Communication Review*, vol. 17, no. 5. ACM, 1987, pp. 2–7.