Tbooster: Adaptive Touch Boosting for Mobile Texting

Nohyun Jung, Gwangmin Lee, Seokjun Lee, Hojung Cha
Department of Computer Science
Yonsei University, KOREA
{nhjung,gmlee,sjlee}@mobed.yonsei.ac.kr, hjcha@yonsei.ac.kr

ABSTRACT
Current mobile devices use a touch boosting scheme to handle operations caused by user interactions with the touchscreen. The current scheme uses a predetermined DVFS step for touch boosting, regardless of user texting speed or related workloads, causing power waste due to unnecessarily high CPU frequency. In particular, the current mechanism is not optimized for power usage when the soft keyboard is used as an input mechanism. In this paper, we propose a scheme called Tbooster, which adaptively adjusts touch boosting level. The scheme reflects texting interval and texting latency, minimizing power consumption while maintaining the user's quality of experience. The scheme was implemented in Android devices and evaluated using a variety of texting applications. Our evaluation results show that the proposed technique reduces the device's overall power consumption by 4.6–13.1%, depending on texting interval and application type.

Keywords
Smartphone; Touch boosting; Soft keyboard; Texting; Power; User experience

1. INTRODUCTION
User inputs in mobile devices typically involve touch events by means of a touchscreen. User-perceived latency (the time to update the result of operations caused by the touch event) is critical and greatly affects the user's quality of experience (QoE) [1, 2]. Touch boosting is a common technique to ensure the required latency, improving QoE by instantly boosting CPU performance when a touch event occurs. This technique optimizes user-perceived latency by overriding the CPU's default DVFS (Dynamic Voltage and Frequency Scaling) governor.

Although touch boosting enhances QoE, the scheme does not always work effectively in terms of power consumption. Touch boosting normally uses a predetermined CPU frequency, regardless of features of touch events such as touch pattern and implied workload. Although this scheme ensures high performance, the required QoE can, in some cases, be achieved with lower performance [3]. That is, the current approach to touch boosting often causes power waste by maintaining an unnecessarily high CPU frequency in some situations.

HotMobile '16, February 26-27, 2016, St. Augustine, FL, USA
 ISBN 978-1-4503-4145-5/16/02$15.00
DOI: http://dx.doi.org/10.1145/2873587.2873592

One opportunity for optimizing current touch boosting arises when users input texts via soft keyboard. The workload due to texting is affected by two factors: (1) type of operations occurring during texting and (2) user texting speed. First, texting applications may incur diverse workloads during texting. For example, in contrast to simply outputting texts on the screen, an auto-complete feature or view update requires a non-trivial amount of workload during texting. Second, various factors such as keyboard layout, typing proficiency, and even user age can affect texting speed, which varies widely among individuals. For slow texting in particular, there are opportunities to reduce power consumption, as the related workload can be loosely processed without degrading QoE. The current approach to touch boosting does not take account of the impact of all these characteristics of texting on power efficiency.

In this paper, we describe a touch boosting system called Tbooster, which adaptively alters boosting level to reduce power consumption for soft keyboard-based texting. Our goal is to reduce power consumption while ensuring QoE by analyzing the characteristics of mobile texting. Our scheme contributes in three respects. First, we experimentally analyze the impact of touch boosting on both performance and power consumption. Second, we introduce and define the concept of Critical Texting Latency, based on an analysis of the relationship between texting latency and QoE. Finally, we optimize the touch boosting system by using an adaptive boosting control that reflects texting interval and texting latency.

2. BACKGROUND
2.1 Touch Boosting Mechanism
Modern mobile devices use the DVFS policy for the efficient management of CPU power consumption. The Ondemand governor or similar is included as a default governor in the operating system to adjust CPU frequency, based on periodic measurement of CPU utilization. For this reason, it is not optimized to handle immediate responses such as touch events. In particular, for touch events that occur at low CPU frequency in the idle state, the policy is unable to provide a satisfactory user experience. To overcome this limitation and to improve QoE for touch events, current mobile devices use a touch boosting mechanism. Figure 1 illustrates how this mechanism is implemented in an Android system. If ACTION_DOWN occurs, the system boosts performance by fixing the minimum DVFS step as defined in the touchscreen driver. If the touch event is maintained for a certain period of time, the minimum step is

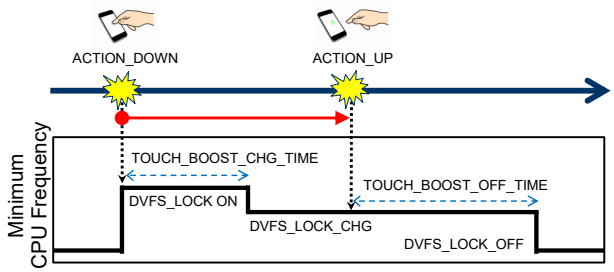

Figure 1. Touch boosting operation in Android system.

lowered. On completion of the touch event by ACTION_UP, the lock is released after a predetermined time has elapsed.

2.2 Implications for Texting

Various kinds of text input mechanisms are available for mobile devices, such as speech-to-text or Swype, but texting via soft keyboard remains the most popular [4]. Recent surveys show that, on average, users send 110 SMS messages per day [5]. Social networking applications are another modality requiring text input.

When text input occurs, the amount of associated workload varies widely, depending on the target application. For simple text output, the workload is light. However, for example, AutoCompleteTextView handle additional work for the autocomplete feature that predicts the rest of the word being typed. Additional workload varies according to the approach of the application developer. User texting speed also affects the workload incurred by texting; for faster texting speeds, the workload per unit time obviously increases. In practice, various factors such as age, keyboard layout, and proficiency can affect a user's texting speed.

3. PRELIMINARY EXPERIMENT

We conducted preliminary experiments to investigate the impact of touch boosting on both power consumption and performance.

3.1 Experimental Setup

To begin, users were surveyed to identify the categories of applications most often used for texting. Sixty people participated in the study. They were selected via Internet posting with brief interviews. 42 are in their 20's and 18 in 30's (33 male and 27 female). 36 of them were college students and 24 were office workers. The participants were using 12 different types of smartphones, and their device use was less than 1 year (33 of them), 1~2 years (16), 2~3 years (9), and more than 3 years (2). 37 of them answered they are proficient with device use, whereas rest of them were found naïvely using their devices.

The survey results are shown in Figure 2 (a). Instant Messaging (IM) applications were used by an overwhelming majority (97%) of respondents; SNS by 45%; memo by 32%; browser by 30%; and e-mail by 18%. Based on these results, ten applications were chosen from five categories (see Figure 2(a)) for the subsequent experiment. Table 1 shows the types of text-input UI (user interface) for each application—for example, Gmail type 1 is the text view used to input an email address and Gmail type 2 refers to content input.

Next, experiments were conducted to measure the general texting speed of the 60 participants. For this purpose, an application was written to log the timestamps of texting events while the participant entered short sentences. Figure 2 (b) shows the

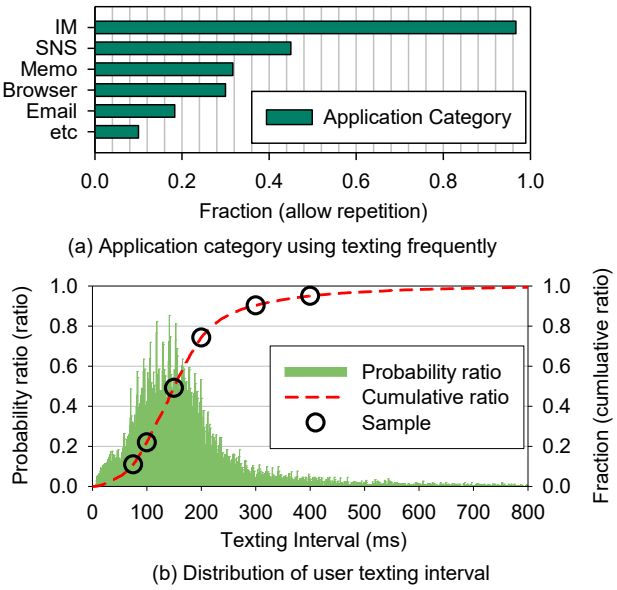

Figure 2. Application category (survey results) and user texting speed (experiment).

distribution of texting intervals for 27,138 events. For one short-sentence input, the average of texting interval varied from 75 ms to 344 ms. Based on this experiment, six different texting intervals were designated: 75, 100, 150, 200, 300, and 400ms; these parameters were used for all subsequent experiments.

The target device used for experimental purposes was the Samsung Galaxy S5 (SM-G900F), running Android 4.4.2 KitKat. The device is equipped with a quad-core Qualcomm Snapdragon 801 processor and provides fifteen DVFS steps, ranging from 0.3 to 2.4 GHz. The device uses the Synaptics touchscreen, which supports touch-boosting. The DVFS step used for touch boosting is the fourth highest (1.7 GHz); 1.2 GHz is used when a touch event lasts longer than 130 ms. DVFS locking is released at 500 ms after the end of touch event. The Interactive governor is the default governor used in the Galaxy S5. It has a similar policy to that of the Ondemand governor, but more heuristics and configurable parameters are provided to enhance response time.

During the experiment, Android Monkey was used to generate a fixed interval for touch events. We noted that touch boosting occurs only if a touch event is processed by the touchscreen driver, and this does not apply to the use of Monkey. To solve this problem, we developed a kernel module specifically for Android Monkey. The module uses kprobes to detect Monkey touch events by checking the IPC data in the binder driver. Depending on the type of touch events, it calls the touch boosting function of the touchscreen driver, enabling operation as general touch boosting. Provisioning of touch boosting was controlled by enabling and/or disabling this module.

Table 1. Type of text-input UI in test applications

Application	Type 1	Type 2
IM: Telegram,WhatsApp	Search	Message
SNS: Facebook,Twitter	Search	Post
Memo: GoogleKeep,NaverMemo	Search	Memo
Browser: Chrome,Firefox	Search	WebSearch
Email: Gmail,Naver Mail	Address	Mail Subject

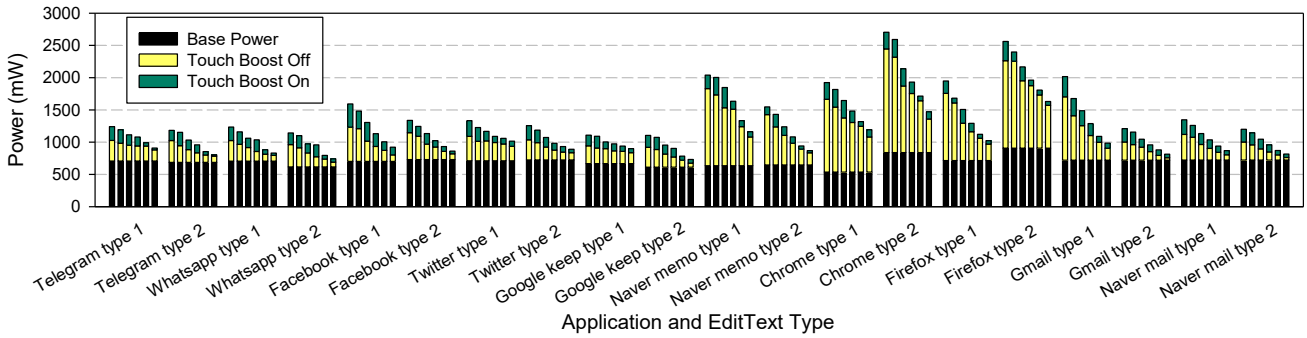

Figure 3. Average power consumption on texting (bars from left to right in each group indicate 75, 100, 150, 200, 300, and 400ms texting intervals).

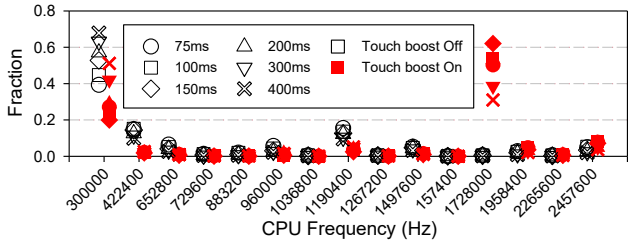

Figure 4. CPU frequency distribution when texting.

3.2 Power Consumption when Texting

To investigate the impact of touch boosting on power consumption, power consumption (with and without touch boosting) was measured while texting, using two different text-input UIs for each of 10 applications. For the experiments, the radio interface was disabled, except for Wi-Fi, which was required to provide a stable Internet connection for the browsers. The display was set to stay awake at 50% brightness, and screen auto-rotation was disabled. The target device's power consumption was measured using the Monsoon Power Monitor.

Average power consumption was measured for 100 text input events occurring at regular intervals; results are presented in Figure 3. For each text-input UI type, the input intervals used were 75, 100, 200, 300, and 400 ms (based on the results of the previous experiment in Section 3.1). In the bar chart, the interval gradually increases from left to right. Base power is the amount spent while waiting for text input. Note that base power differs for each UI type because of the characteristics of the OLED display, whose power consumption depends on usage of colors.

Figure 3 shows that power consumption increases as texting interval is shortened. This is because unit workload increases as touch events occur more frequently. There is also a difference in power consumption for each input type, as the amount of work related to texting varies with the application. For example, Gmail type 1 is AutoCompleteTextView, which requires additional work for the auto-complete feature and therefore consumes more power than Gmail type 2, which simply processes text input. Overall, without touch boosting, the device's power consumption was reduced by about 11.6% on average for the boost-on case. This is equivalent to a power reduction of 31.8% (without considering base power). The results indicate that the current approach to touch boosting significantly affects power consumption when users input texts by means of a soft keyboard.

Power consumption is caused primarily by the high DVFS step used for touch boosting. We measured CPU usage per frequency in the same environment as in the previous experiment; Figure 4

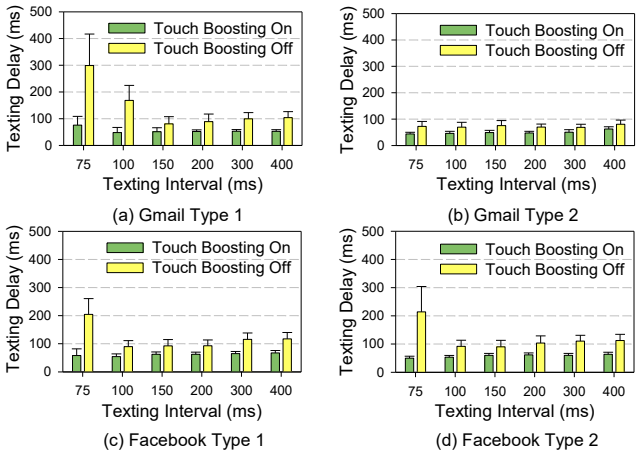

Figure 5. Texting latency versus texting interval.

shows the distribution of CPU usage. When touch boosting is disabled, the Interactive governor controls CPU frequency on the basis of CPU utilization, regardless of the touch event. The recorded usage of 0.3GHz (the lowest step) was overwhelmingly high. Usage of 1.2GHz was the second largest, but accounted for less than 20%. On the other hand, with touch boosting enabled, CPU frequency was boosted on user's touch. Since texting rapidly generates ACTION_DOWN and ACTION_UP, the DVFS_LOCK_CHG frequency is not used here. Instead, only the DVFS_LOCK frequency is used for touch boosting, and so the DVFS_LOCK frequency of 1.7GHz used by touch boosting was most frequently used; 0.3GHz came second. This result shows that enabling touch boosting maintains higher performance, resulting in higher power consumption. As touch boosting uses only one CPU frequency, this represents a good opportunity for optimization.

3.3 Texting Performance

As the purpose of touch boosting is to improve QoE, the analysis of its impact on system performance is critical. Experiments were conducted to measure texting latency (the time delay before input text is displayed on the screen) for the Gmail and Facebook applications; Figure 5 shows the results. With touch boosting enabled, texting latency varies from a minimum 43.8 ms to a maximum 75.9 ms, with an average of 56.3 ms. Texting latency does not change significantly even when texting interval is altered. However, with touch boosting disabled, texting latency is greatly affected by the texting interval. Since the power governor monitors workload periodically, there is a delay in altering CPU frequency, during which CPU performance may not be adequate

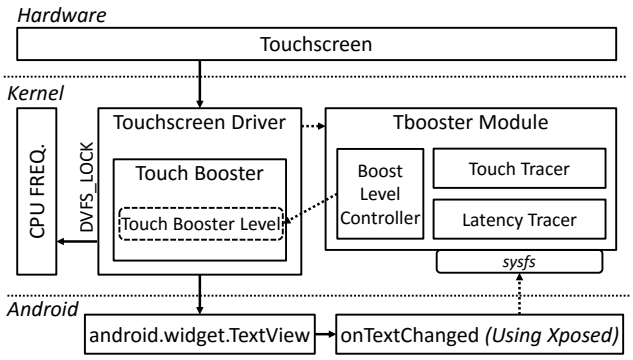

Figure 6. Overview of Tbooster.

for the workload. It follows that when the texting interval is very short, the delay increases because of the high workload. For a long texting interval, texting latency increases only slightly because of the small workload. Furthermore, Gmail type 1 and type 2 behave differently when touch boosting is disabled; Gmail type 1 shows a large increase in delay as texting interval decreases while Gmail type 2 does not. For Gmail type 2, texting delay also remains more or less unchanged with or without touch boosting. That is, for Gmail type 2, touch boosting maintains unnecessarily high performance and does not need any boost, as simple text viewing incurs only a very small workload in this case.

From this experiment, it can be observed that the effect of touch boosting on user experience varies according to the workload that accompanies texting, making it necessary to vary CPU frequency for touch boosting in different texting situations.

4. TBOOSTER

This section describes an adaptive touch boosting scheme that dynamically adjusts touch boosting level. An explanation of the overall architecture is followed by a discussion of critical texting latency.

4.1 System Overview

To optimize power consumption while providing adequate performance when texting via soft keyboard, we designed and implemented an adaptive touch boosting system called Tbooster. Unlike the current approach to touch boosting, which uses a predetermined CPU frequency, Tbooster alters CPU frequency dynamically, according to the texting situation. Figure 6 illustrates the overall architecture of the scheme. The system consists of three key components: Touch Tracer, Latency Tracer, and Boost Level Controller.

Touch Tracer collects texting events and timestamps for texting events. The texting interval is calculated using the timestamps, and the average recent texting interval is subsequently updated. The component uses kprobes to hook *binder_transaction*, which is the kernel function used in Android for inter-process communication. Every command transferred by binder has its own code. When a user input text via soft keyboard, we captured the command and used the code in the command to distinguish ACTION_UP and ACTION_DOWN.

Latency Tracer records the latency required to process text input for output. The component utilizes Xposed to hook the method calls and to control pre- and post-method calls in the Android platform. It hooks *onTextChanged*, which is the function of Android TextView, to obtain a timestamp when text on the screen

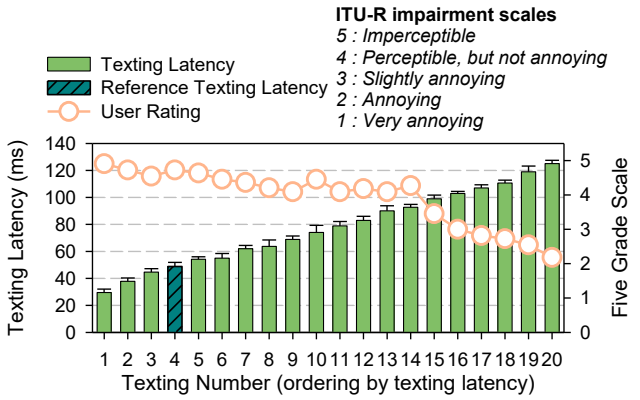

Figure 7. User QoE vs. texting latency.

is changed. Based on the acquired timestamp, texting latency is then calculated.

Boost Level Controller determines the boosting level on the basis of information from both Touch Tracer and Latency Tracer. The *critical texting latency* can then be determined by comparing texting interval and texting latency for the last five touch events or for the last second (see Section 4.2. for more details). The touch boosting level of the touchscreen driver is altered by comparing critical texting latency with recent texting latency. When touch latency is greater than critical texting latency, a high level of touch boost is used (and vice versa).

4.2 Critical Texting Latency

We now define critical texting latency—that is, the texting latency required to ensure QoE during texting. The texting latency should be maintained shorter than critical texting latency.

We conducted a user study with sixteen participants to determine the effect of texting latency on QoE. For this study, CPU frequency was fixed at maximum to maintain constant performance. For the purposes of the experiment, we developed a simple application and asked users about their level of satisfaction with touch latency. In the application, twenty types of button were provided for random assignment of texting latency from 30 to 125ms. The default button had a texting latency of 50 ms, and this was used as the reference point. The testing methodology involved a double-stimulus impairment scale method, using a five-grade impairment scale as described in ITU-R BT Rec. 500 [6]. Figure 7 presents the results.

When texting latency was between 30 and 100 ms, user ratings decreased slightly, but users did not experience discomfort. The rating changed rapidly when texting latency exceeded 100 ms, as users felt uncomfortable with a texting latency above this level. A recent study [3] that classified workloads in terms of event intensity and latency found that the acceptable QoE boundary for texting is 100 ms for low event intensity (fewer events per second) and low latency (shorter execution time). Our results are consistent with those findings.

In some cases, critical texting latency should be applied conservatively. If texting latency exceeds the texting interval, the next text input will occur before completion of the previous texting operation. Because of these nested operations, texting latency can suddenly increase. Figure 8 shows the result of an experiment to measure texting latency according to touch boosting level with a texting interval of 75 ms. Boost level 12 is the CPU frequency step used in the current touch boosting system, and

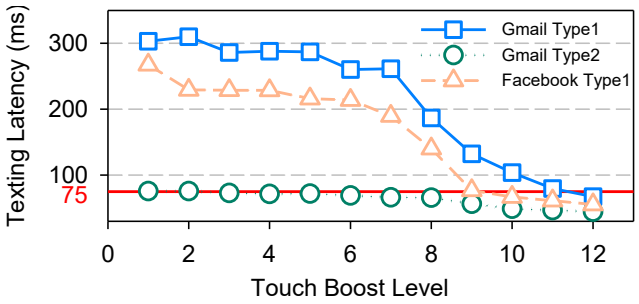

Figure 8. Relation between texting latency and boost level.

boost level 1 is the lowest CPU frequency step. If texting latency is less than the texting interval fixed at 75 ms, texting latency increases slightly even though the CPU frequency used for touch boosting is low. On the other hand, if texting latency is greater than the texting interval, texting latency increases rapidly as the CPU frequency used for touch boosting decreases. Texting interval is therefore used as the critical texting latency in this situation to prevent a rapid increase in texting latency.

Based on the results of the above experiments, a simple algorithm was developed to determine critical texting latency. When the average texting interval exceeds texting latency, user-perceived latency of 100 ms is used as the critical texting latency. Conversely, when the average texting interval is less than the texting latency, texting interval is used as the critical texting latency. Here, user-perceived latency is the latency at which users began to experience discomfort in the previous user study. Based on this critical texting latency, we adaptively control the touch boost level to provide adequate QoE for diverse workloads.

5. EVALUATION

Tbooster was evaluated to validate its effectiveness for power saving, texting quality, and overhead as implemented on a Samsung Galaxy S5 running Android KitKat.

5.1 Power Saving

Experiments were conducted to measure power consumption both before and after applying Tbooster. The 20 texting cases used in Section 3.2 were benchmarked for the purposes of the evaluation. Figure 9 shows the overall effect of power reduction as compared to the current touch boost approach. Power saving effectiveness was found to be dependent on texting interval. As Figure 9 shows, the shortest touch interval of 75 ms leaves little room to lower the touch boost level, as workload per unit time is high and the critical texting latency is 75 ms. Thus power saving at 75 ms is reduced by only 4.6%, with an average of 74.8 mW. Second, for 300, and 400 ms, the power savings are 7.2 and 5.9%, respectively. As shown in Figure 3, touch boosting has less impact

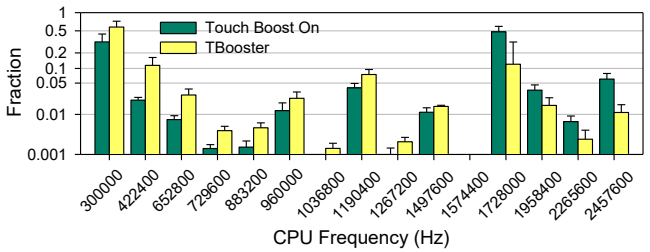

Figure 10. CPU frequency distribution using Tbooster.

on power consumption as texting interval increases, and the reduction of power consumption is small, with averages of 72.6 and 57.3 mW. Finally, at 100, 150, and 200 ms texting intervals, power savings of 11.8, 13.1, 10.8%, respectively, were observed. This is meaningful because, according to the user study on texting intervals in Section 3.1, these account for most of the distribution of users' texting interval. In this range, texting latency is smallest when touch boost is turned off, representing a good opportunity to lower CPU frequency for effective touch boosting. The average power saving for the entire texting interval is 8.9%, and the effect of touching boosting itself (excluding base power) is approximately 25.6%.

To further analyze the power saving effect, experiments were conducted to measure CPU usage by frequency. Figure 10 shows the distribution. Usage of 1.7 GHz (the frequency used for current touch boosting) is considerably reduced by the use of Tbooster. In the case of frequencies lower than 1.7 GHz, CPU usage increases in every case while decreasing for frequencies higher than 1.7GHz; this is because touch boosting level was adaptively adjusted on the basis of texting latency and texting interval. CPU usage, which was concentrated at 0.3GHz and 1.7GHz, is now seen to be evenly distributed across other CPU frequencies.

The above evaluation was done in controlled environment for the exact analysis of the efficacy of the proposed mechanism. In fact, many factors in real use scenario influence the energy consumption of device, and mobile texting would constitute a fraction of the overall portion. Here, one critical question would be how much does this optimization improve day-to-day battery life? The answer is hard to get since the overall energy savings should be measured for typical users under long-term and duplicative use scenarios. Leaving this for future work, we in this work conducted a rough evaluation on the gross effect of Tbooster, with ten apps (Figure 9) plus two more messenger apps - Kakao Talk and Line. Each application was used for 30 minutes under *normal application use*, i.e., no controlled setup, and the energy consumption was measured with and without Tbooster. We observed that during this period device energy is saved 1.9~13%, and 6% on average. As predicted, WhatsApp, Kakao Talk, and Line were most effective since text input is the primary

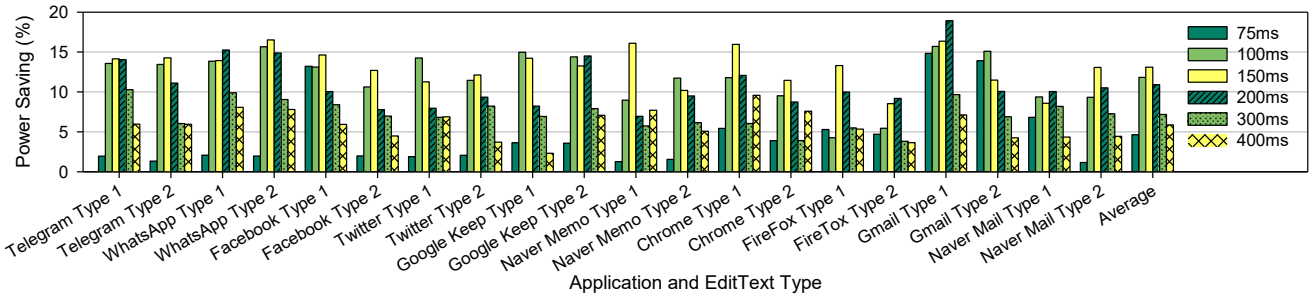

Figure 9. Effect of Tbooster on power consumption.

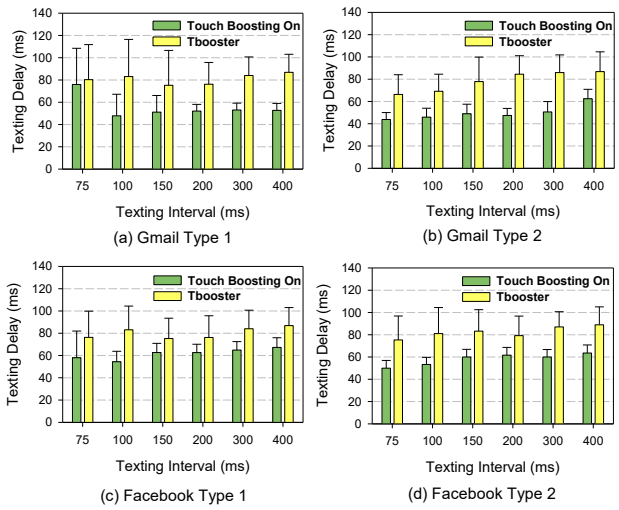

Figure 11. Tbooster: texting latency vs. texting interval.

functionality of the applications. Considering that typical users are frequently using messenger apps for a considerable amount of time in daily basis, we believe that the effect of Tbooster should be meaningful in real world scenarios.

5.2 Texting Quality

Texting latency was measured to investigate any possible degradation of texting quality by Tbooster. Figure 11 shows that for current touch boosting, texting latency is about 55.7 ms, which is far shorter than the acceptable QoE measure of 100ms. Using Tbooster, the average texting latency is 79.6 ms, an increase of 43% above the baseline. However, Tbooster maintains an average texting latency lower than the critical texting latency when texting interval is increased from 100 ms to 400 ms. At 75 ms, texting latency is slightly greater than the critical texting latency, but because it is still lower than the acceptable QoE measure, the quality of user performance is not significantly reduced.

5.3 Overhead

For overhead analysis, we measured the additional power consumption when using Tbooster. For the experiment, the Boost Level Controller was deactivated to render touch boosting identical with that of the default state. Power consumption with the deactivated Tbooster was then compared to current touch boosting. At texting intervals of 75, 100, 150, 200, 300, and 400ms, there was additional power consumption of 0.93, 0.62, 0.33, 0.22, 0.14, and 0.13%, respectively. When the texting interval was shortened, overhead increased because the computation overhead for Tbooster increases as touch events occur more frequently. Nevertheless, even at the shortest texting interval of 75 ms, the power overhead for Tbooster was 0.93%, which is negligible.

6. RELATED WORK

Recent studies have considered user experience and user interaction in relation to managing the power consumption of smartphones. SmartCap [7] showed that a neural network-based inference model is useful in deciding minimum acceptable frequency without degrading user experience. The technique proposed in AURA [8] effectively determines CPU frequency for each interactive session by classifying applications according to intensity of user interaction. A power optimization framework based on user-perceived response time analysis was proposed in [2], and [4] characterized and compared energy consumption for three text input modalities: type, talk, and Swype. In similar vein, the present study investigated touch boosting for soft keyboard input and suggested improvements.

7. CONCLUSION

Current touch boosting systems use a predetermined CPU frequency step that takes no account of factors such as user texting speed, types of texting view, or workloads associated with texting applications. This commonly results in unnecessarily high performance demands, and as a consequence, power is wasted. Here, we have proposed an adaptive touch boosting system for texting that dynamically adjusts touch boosting level on the basis of texting latency and texting interval. Tbooster has been shown to reduce power consumption during texting while maintaining texting latency below critical texting latency. Using real applications, our evaluation shows that Tbooster reduces overall device power consumption by 4.6 to 13.1% (depending on texting interval) while satisfying user-perceived latency.

8. ACKNOWLEDGEMENTS

This work was supported by Samsung Research Funding Center of Samsung Electronics under Project Number SRFC-TB1503-02.

9. REFERENCES

[1] Ravindranath, L., Padhye, J., Agarwal, S., Mahajan, R., Obermiller, I., and Shayandeh, S. 2012. AppInsight: Mobile App Performance Monitoring in the Wild. *OSDI* 12, 107-120.

[2] Song, W., Sung, N., Chun, B. G., and Kim, J. 2014. Reducing energy consumption of smartphones using user-perceived response time analysis. *Proc. 15th Workshop on Mobile Computing Systems and Applications*.

[3] Zhu, Y., Halpern, M., and Reddi, V. J. 2015. Event-based scheduling for energy-efficient QoS (eQoS) in mobile web applications. *Proc. IEEE 21st International Symposium on High Performance Computer Architecture (HPCA)*.

[4] Jiang, F., Zarepour, E., Hassan, M., Seneviratne, A., and Mohapatra, P. When to Type, Talk, or Swype: Characterizing Energy Consumption of Mobile Input Modalities.

[5] Nielsenwire. 2010. US teen mobile report: Calling yesterday, texting today, using apps tomorrow [blog post]. http:/fblog. nielsen. com/nielsenwire/onlirte_rnobile/us-teen-mobile-report-calling-yesterday-texting-today-using-apps-tomorrow.

[6] BT.500-13 ITU-R Recommendation, "Methodology for the subjective assessment of the quality of television pictures", *Tech. Rep. BT.500-13*, ITU, 2012.

[7] Li, X., Yan, G., Han, Y., and Li, X. 2013. SmartCap: user experience-oriented power adaptation for smartphone's application processor. *Proc. Conference on Design, Automation and Test in Europe*.

[8] Donohoo, B. K., Ohlsen, C., and Pasricha, S. 2011. AURA: An application and user interaction aware middleware framework for energy optimization in mobile devices. *Proc. IEEE 29th International Conference on Computer Design (ICCD)*.