# Kernel Level Energy-Efficient 3G Background Traffic Shaper for Android Smartphones

Ekhiotz Jon Vergara, Joseba Sanjuan, Simin Nadjm-Tehrani
Department of Computer and Information Science
Linköping University, Sweden
{ekhiotz.vergara, joseba.sanjuan, simin.nadjm-tehrani}@liu.se

***Abstract*—Reducing the energy consumption of wireless devices is paramount to a wide spread adoption of mobile applications. Cellular communication imposes high energy consumption on the mobile devices due to the radio resource allocation, which differs from other networks such as WiFi. Most applications are unaware of the energy consumption characteristics of third generation cellular communication (3G). This makes the background small data transfers of undisciplined applications an energy burden due to inefficient utilisation of resources.**

**While several approaches exist to reduce the energy consumption of this best-effort background traffic by means of traffic shaping, we find that they are mostly evaluated with simulations and the actual energy overhead for the traffic shaper itself has not been studied. In order to cover this gap, our work realises an existing energy saving algorithm as a Kernel Level Shaper (KLS) within the Android platform, and measures its energy footprint. The total energy savings of our implementation range from 8% to 58% for emulated real background traffic, that is categorised as best-effort traffic. We further show the implications of running the KLS during live operation of applications as an exploratory study.**

***Index Terms*—transmission energy; 3G; kernel; Android**

## I. Introduction

Mobile users have been blessed with ubiquitous connectivity and powerful devices with the advances in wireless technologies, but they still experience short battery lifetimes which makes energy consumption the Achille's heel of user quality of experience. The massive mobile data traffic explosion and its forecasted tremendous growth further aggravates the situation.

While the device manufacturers are busy making the device hardware more energy-efficient, the lack of energy awareness on the software front is becoming all too apparent. Those application developers that care about efficiency of their applications are more likely to address responsiveness, or perhaps to test the energy efficiency in a WiFi setting. However, *true* mobility is best handled by the cellular mode of communication.

Cellular networks impose high energy consumption on the mobile devices due to the radio resource allocation performed at the operator end, which differs from other technologies such as WiFi. Most applications are unaware of the transmission energy characteristics of cellular networks, where the energy consumption is not proportional to the amount of data sent: small data transfers can consume as much as a fully utilised link using 3G. Moreover, we find that even without user interaction, undisciplined transmissions are performed by applications inefficiently utilising the limited energy resource. In particular, the best-effort background traffic created by applications (e.g., periodic transfers or updates) leads to high energy consumption, even though it is only a small part of the total traffic.

There exist several approaches [1]–[8] to potentially reduce the energy consumption of these cellular data transfers that do not have any quality of service requirements. Most of these rely on shaping the traffic. Some approaches aggregate and send together all transfers resulting in higher delay, or align periodic transfers of different applications to perform the data transfers within the same interval. However, as we describe in the related works section, most approaches are evaluated through simulation using pre-recorded packet traces and do not consider applications' live operation and application-protocol interactions. Moreover, the energy consumption of the solution itself is not analysed.

Our work covers that gap by developing a prototype of an energy saving algorithm on a commodity device and evaluating its operation in a real environment. The contributions of our work are: (a) realization of an existing algorithm (from [1]) on the Android platform, (b) measuring the energy footprint of the algorithm operation, and (c) discovery of the implications of using such a solution with real traffic generated during applications' operation (which cannot be steered towards the sort of traffic that it was designed for), thereby identifying future directions of research.

The rest of the paper is organised as follows: section II provides the energy consumption background from 3G and the energy reduction of background traffic. The architecture of the traffic shaper is described in section III and evaluated in section IV. Section V describes the related works, and section VI concludes the paper leading to future work.

## II. Background

This section provides a background on Universal Mobile Telecommunications System (UMTS) energy consumption at the user end. This illustrates the implications of the energy consumption of the background traffic, and reviews the energy saving algorithm that is used in the paper.

978-1-4673-2480-9/13/$31.00 

### A. Energy consumption of 3G

The energy consumption of the user equipment (UE) in 3G is mostly influenced by the state machine defined by the Radio Resource Control (RRC) protocol at the cellular operator end. The Radio Network Controller (RNC) uses the RRC protocol and the Radio Link Control (RLC) protocol to perform the radio resource management of the UE.

The UE experiences different power drain and performance in terms of throughput and response time depending on its state. The UE states are Dedicated Channel (DCH), Forward Access Channel (FACH), and Paging Channel (PCH), sorted in terms of highest to lowest power drain and performance.

The transitions between states are controlled using inactivity timers, and the traffic volume measurements reported by the UE using the RLC protocol. The RNC employs fixed thresholds over the RLC buffer data occupancy to trigger state transitions to higher performance states (PCH-FACH or PCH-DCH and FACH-DCH). Four thresholds (2 uplink and 2 downlink) control the different transitions. A transition occurs when the data occupancy exceeds a given threshold.

Inactivity timers $T_1$ and $T_2$ are used to downswitch the UE to lower performance states when there is small or no data transmission. $T_1$ controls the DCH-FACH transition whereas FACH-PCH is controlled by $T_2$. Inactivity timers force the UE to stay in a high power state while not transmitting anything, creating energy overheads known as energy *tails* [2], [9].

The upper part of Fig. 1 shows an illustrative example of the energy consumption of an application as measured by a 3G broadband module provided by Ericsson AB while switching between the different states. The UE is forced to move from FACH to DCH in the beginning. Then it switches back to FACH after $T_1$, and to PCH after $T_2$ (before 20 s).

### B. Energy consumption of background traffic

The Quality of Service (QoS) of the third generation UMTS standard specifies the *background traffic* class. This traffic class is usually characterised by low bandwidth requirement, intermittent and asymmetric (uplink/downlink) data transmissions and a permissible delay since the destination does not expect the data within a certain time (c.f., RSS or e-mail). Background traffic is categorised as elastic and best-effort [10].

Mobile applications create background traffic even when the mobile device is not being used (e.g., screen switched off [11]). Even though the volume of these background transmissions is not large, this best-effort traffic is significant enough due to the energy footprint of 3G. The upper part of Fig. 1 shows an example of energy consumption of background transmissions by two commonly used applications (Skype and Facebook), which forces the UE to move even to DCH.

Earlier works have proposed the reduction the energy consumption of background traffic based on the intuition that applications can defer their best-effort data transmissions [2], [12], [13]. These transmissions are aggregated and sent in a single burst. In particular, the Cross-Layer Burst Buffering (CLBB) algorithm [1] schedules background uplink traffic in an energy-efficient manner using this general idea.

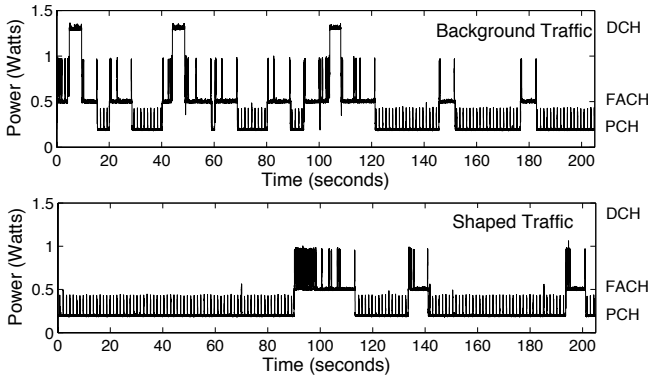


Fig. 1: Energy consumption examples of background traffic and energy-efficient shaped background traffic.

Given a maximum waiting time for a packet transmission ($T_w$), CLBB uses the knowledge of inactivity timers and RLC buffer thresholds to perform data aggregation: when the UE is in PCH, CLBB does not send packets until their $T_w$ is about to expire, maximising the time in the lowest consuming state (PCH) and reducing the number of energy tails. The packets are sorted by nearest (latest) transmission time. When a group of packets needs to be sent, CLBB schedules small data packets in the FACH state avoiding unnecessary and energy-costly state transitions to DCH. This is done by classifying the packets in two categories (small and large) based on the RLC buffer thresholds and the packet size. When buffered packets cannot be sent in FACH due to their size, CLBB sends the data in a single burst moving the UE to DCH as previous works [2], [12], [13]. Fig. 1 shows an illustrative example of the energy consumption of background traffic and the traffic scheduled by CLBB. In this example, CLBB reduces the energy consumption of the background traffic (when $T_w = 90$ s) by 34%.

## III. Architecture design

The CLBB algorithm schedules background traffic to reduce the 3G energy consumption of the UE. The operation of the CLBB implementation should be as energy-efficient as possible so that the energy savings achieved by the algorithm are not diminished. In this section we describe the design of the Kernel Level Shaper (KLS) architecture implementing the CLBB algorithm in the Android platform.

Intercepting and modifying the traffic transmissions generated by applications can be performed in different layers of the Android architecture and the Linux networking stack. The architecture of the KLS resides in kernel space. A user space application would require to interact with the kernel space through system calls, which would create a higher CPU footprint due to data transfer (read/write). A single data transfer can result in a CPU usage up to 33% [14]. Moreover, the throughput in kernel space is higher [15].

In the Linux kernel, a networking packet is represented using the socket buffer structure (*sk_buff*), which is sent through the layers of the networking stack until it is sent by

the networking card. Netfilter is a Linux kernel framework that enables packet interception and modification at different *hooks* (decision points in the networking stack). Kernel tampering is another alternative for packet interception, which consists of modifying the code of the native kernel functions that control the packet traversal flow. Kernel tampering provides higher TCP/IP throughput than Netfilter, but the background traffic of applications does not have high throughput requirements. Netfilter provides modularity and scalability by using a Kernel Loadable Module (LKM) that can be inserted at runtime, instead of changing the native code at kernel compiling time. Therefore, KLS implements CLBB as a LKM registering the Netfilter *NF_IP_POST_ROUTING* hook in the IP layer. This allows us to intercept the outgoing packets without interacting with applications, leading to low CPU footprint.

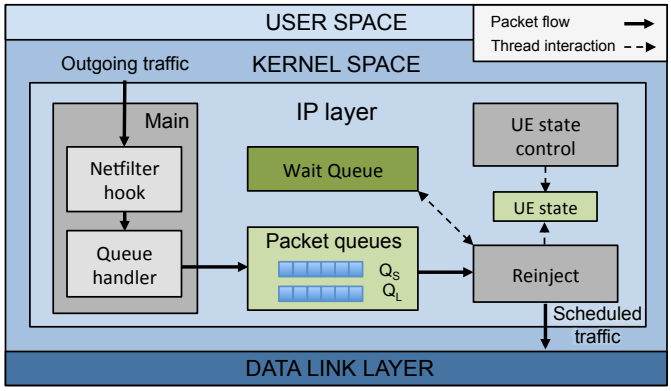


Fig. 2: Overview of the Kernel Level Shaper architecture.

The architecture of the KLS shown in Fig. 2 consists of three threads (Main, Reinject and UE state control). The Main thread registers the hook, starts the other threads and categorises the outgoing packets in the queues based on size ($Q_S$ for small packets and $Q_L$ for large ones) and sorted by nearest $T_w$. The Wait queue is a Linux mechanism used to sleep or wake up the Reinject thread when there are no packets. The Reinject thread transmits the queued packets following the logic of the CLBB algorithm and updates the UE state based on the volume of data transmitted. The UE state control thread keeps track of the UE state using inactivity timers and the data transmissions. Thread concurrency is achieved by means of the kernel lock mechanism.

## IV. Evaluation

This section evaluates kernel level shaper, implementing the CLBB algorithm embedded in an Android device. It focuses on two main aspects: (1) the validation of the 3G energy-savings of CLBB in the mobile device, and (2) analysing the energy footprint of the implemented energy saving algorithm. The KLS architecture is evaluated for traffic with different characteristics. We describe our methodology in the next section.

### A. Methodology and experimental setup

The architecture is deployed in a rooted Android device (HTC Sensation Z2710e). We cross-compile the Linux kernel with the needed options (e.g., enable module loading support and Netfilter), boot the new kernel in the device, cross-compile our KLS and load it at runtime.

Our traffic generation approach is divided in two parts: emulated and live traffic. The choice of the emulated traffic is motivated by the need to create the same application traffic in various experiments. In order to provide experiment repeatability and control over the traffic generation of applications, real background packet traces were captured and replayed with and without the KLS in the Android device. We call this emulated traffic since we replay previously captured real packet traces. These traces contain best-effort traffic that have no specific QoS requirements. Next, as an exploratory phase, we evaluate the algorithm implementation employing the KLS with applications that create live traffic.

For the emulated traffic, *tcpdump* and an *iptables* based firewall were used to capture only the traffic from the chosen applications. Facebook, Skype and WhatsApp were used as test applications and all the traces were recorded with the screen switched off and without any user interaction. These applications created different traffic load over time based on their operation. Since the energy saving of the algorithm depends on the amount of traffic, out of the used traces we selected what we believe are representative traces for different traffic volumes and characteristics. The packet trace is replayed from an Android application sending uplink UPD packets to a server that replies with the downlink traffic. The traces are around 10 minutes long. Fig. 3 shows packet size, inter-packet interval (IPI) and the traffic volume characteristics of the traces. The name of the traces correspond to their traffic volume (low, medium and high). The traces have a majority of small packets and the transmissions range from regular (high) to sporadic (low) as the IPI shows.

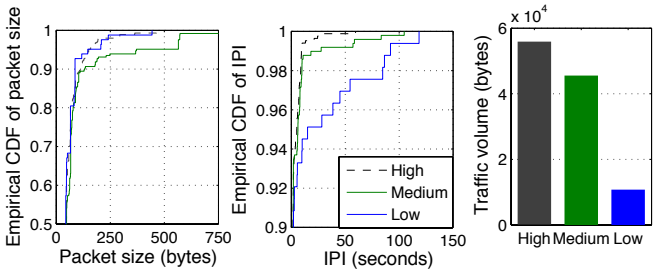


Fig. 3: Characteristics of the emulated traces.

The traces described in Fig. 3 are used varying the maximum waiting time for a packet transmission ($T_w$) of the algorithm (30, 90 and 180 s). For every trace, we replay the trace without KLS as baseline. Then, the KLS is loaded and the traces are replayed for the different $T_w$. We measure the energy savings and CPU usage of the KLS as described below.

**Networking energy consumption:** the transmission energy consumption is calculated using EnergyBox. Given the 3G

network parameters specified at operator level, EnergyBox derives the 3G states of the UE employing trace-based iterative packet-driven simulation. The total energy consumption is calculated by associating the UE specific power levels with the emulated intervals in each state, and integrating them over time. EnergyBox has been evaluated against physical energy consumption measurements showing an average accuracy of 98% [16]. The 3G network settings we use correspond to the operator TeliaSonera in Sweden measured in our local area, which are: $T_1$ = 4.1 s, $T_2$ = 5.6 s, the RLC buffer uplink (PCH-DCH = 1000 and FACH-DCH = 294 bytes) and downlink thresholds (PCH-DCH = 515 and FACH-DCH = 515 bytes) [1]. We set the following power values for the different 3G states based on measurements [9]: DCH = 600 mW, FACH = 400 mW and PCH = 0 W. By setting PCH = 0 W, we aim to quantify only the energy spent in data transmission and energy tails.

**CPU energy consumption:** the CPU usage of the KLS itself is retrieved using the *top* utility. We perform physical measurements in the HTC Sensation to characterise the power usage for different CPU loads in a simplistic manner. The measurements are performed by replacing the battery of the smartphone by a low-side sensing circuit [17]. The frequency of the CPU is set to the maximum (1.2 GHz) in order to measure the highest power level, i.e., worst case. A CPU stress application is developed to increase the CPU load step by step up to 100%. The CPU power is isolated from other factors such as screen, network interfaces or other applications by switching the components off and killing all the processes that can interfere in the measurement. By interpolating the (CPU load, power) points we obtain a function that is employed to obtain the power consumption out of the CPU load data.

The next section describes the results of the evaluation by employing the previously described methodology and setup.

### B. Energy-savings of network transmission

In this section we validate the energy savings of the KLS in the HTC Sensation for the 3 representative background traffic traces (High, Medium and Low) described in section IV-A. We present the energy savings of the UE with the embedded implementation of the CLBB algorithm. The replay of the captured trace without employing KLS is referred to as *original traffic* and used as the baseline.

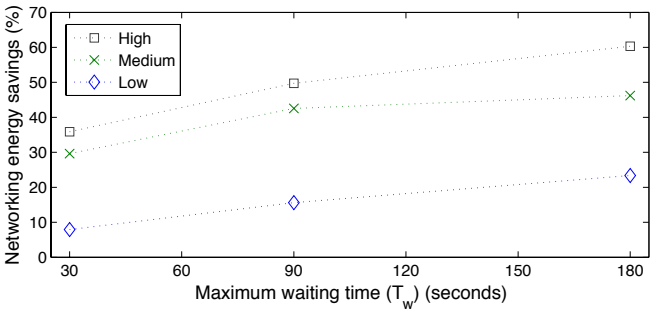


Fig. 4: Energy savings of networking transmissions.

Fig. 4 shows that the energy savings depend on the traffic volume: when the traffic volume is higher, the energy savings tend to be higher. The original traces consume 225, 161 and 105 Joules for the high, medium and low traffic volume traces respectively. Moreover, as expected, increasing $T_w$ leads to higher energy savings. For the low traffic volume trace, the energy savings range from 8% ($T_w$ = 30 s) to 23% ($T_w$ = 180 s). However, the energy savings are much greater (35% when $T_w$ = 30 s) with more frequent background traffic. The maximum registered is 60%.

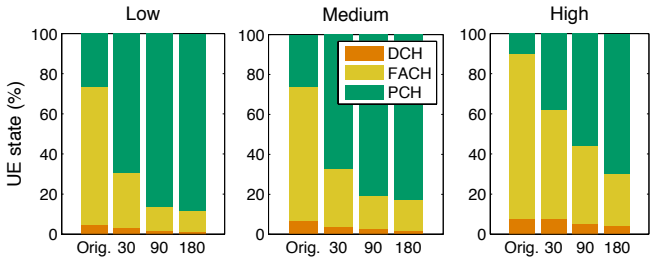


Fig. 5: Percentage of time spent by the UE in the different 3G states over the total time of the original traces (y axis) scheduled with different $T_w$ (seconds) by the KLS (x axis).

Fig. 5 shows the percentage of time spent by the UE in the different 3G states over the duration of the original traces compared to the case where the traffic is scheduled by the KLS. The KLS extends the time spent in PCH state which leads to energy savings. Intuitively, a greater $T_w$ makes the UE to be in PCH for longer period. In general, the amount of time in FACH is reduced in all the cases. For the original traces, it can be noticed that the UE spends more time in FACH with higher amount of traffic. Regarding DCH, the UE performs occasional burst transmissions with the High and Medium traffic profile, leading the UE to DCH for a short time period. However, with Low traffic profile, the UE rarely transitions to DCH.

To summarise, we validate that the KLS implementation of the CLBB algorithm provides high energy savings to the UE.

### C. Energy footprint of the KLS

In this section we show the energy cost of running the algorithm implementation. The KLS performs two main operations, intercepting and queueing the packets, and sending them (dequeueing and re-injecting them in the network stack). The rest of the time the threads of the KLS are sleeping and therefore do not waste CPU time and energy.

In order to quantify the energy footprint when queueing packets, we queue several UDP packets from an application and observe the CPU load created by the KLS. The CPU usage of the KLS is logged and the energy footprint is obtained from the CPU load employing our CPU power characterisation explained in section IV-A. Our measured CPU load is below 0.01% which shows that the footprint due to the queuing operation is negligible in terms of CPU and therefore energy consumption.

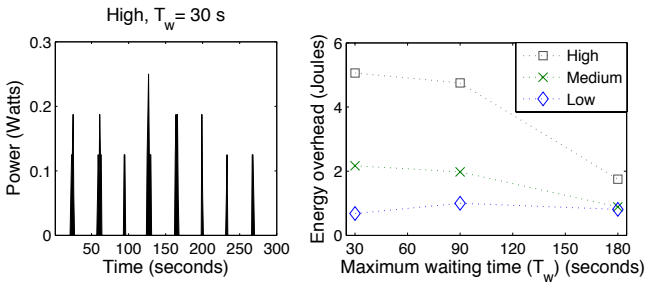


Fig. 6: Example of CPU energy consumption by running the KLS (left) and CPU energy footprint for the different traces (right).

Next, the CPU energy footprint due to sending the packets is measured for the high, medium and low traffic volume traces. Fig. 6(left) shows an example of the energy consumption when running the KLS for $T_w$= 30 s for the high traffic volume trace. The KLS only creates CPU load spikes that are lower than 5%, which consume less than 0.25 W as it is shown in Fig. 6. These power spikes do not consume much energy.

Fig. 6(right) shows the additional energy consumption of the CPU for the traces varying $T_w$. The maximum energy cost is 5 Joules, which is only 2% of the energy consumed for sending the original high traffic volume trace (225 Joules as shown in section IV-B). The energy footprint is higher for traces with higher traffic volume since the KLS has to process more packets. Except for the case of low traffic volume, a greater $T_w$ leads to lower energy consumption.

To summarise, we see that the maximum energy footprint depends on the amount of traffic and $T_w$, but it is low compared to the energy used to send the original traffic due to energy-efficient kernel implementation of the algorithm.

### D. Total energy savings

This section presents the total energy savings, i.e., the network transmissions savings minus the energy cost of the KLS, i.e., the overhead. Since the energy cost of the KLS is small, the energy savings still remain high.

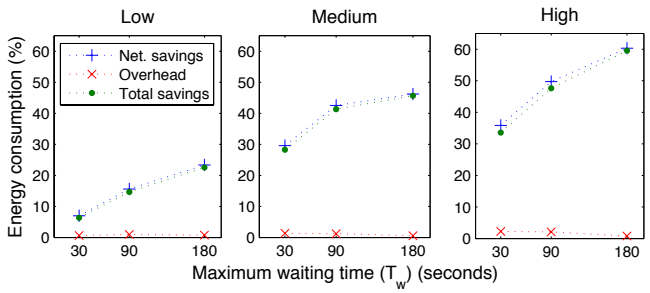


Fig. 7: Network transmissions energy savings, KLS energy overhead and total energy savings as a percentage of no-KLS base energy consumption.

Fig. 7 shows the networking savings of the KLS, the energy cost of the KLS and the total energy savings as a percentage of the total energy consumption of the original traces (255, 161 and 105 Joules for High, Medium and Low traces respectively). When the traffic is low or moderate, the overhead is low and the total energy savings remain close to the networking energy savings. A somewhat higher energy overhead is observed for the High data profile, but still the total energy saving is significant. In this case, the highest saving is achieved with the highest $T_w$, leading to a 58% net saving.

### E. Live traffic

Applications do not distinguish their background traffic from the rest of the traffic, neither do they specify which traffic could potentially be shaped or delayed. This results in the KLS (or a similar approach) scheduling all outgoing traffic. As an exploratory phase, we test the KLS in the current settings to study the implications of performing traffic shaping during live operation of the same applications.

Facebook and Skype are used as test applications with and without the KLS ($T_w$=30 s). We capture traces with and without the KLS to compare the standard traffic behaviour against the behaviour with the KLS. The screen of the smartphone is switched off and we do not actively use the device.

**Skype:** the UE experiences a significant increase in traffic when running Skype with the KLS (51 packets without the KLS against 1145 packets with the KLS). We identify several factors that explain the increase of the traffic, which are dependent on the application's live operation or behaviour.

The Skype client periodically builds a host cache (IP address and port) of Skype servers (supernodes) using the DNS protocol. The Skype client sends a UDP packet of 18 bytes to the port 33033 of a supernode on its host cache. When there is no response for 5 seconds, the client attempts to establish a TCP connection. In case of failure, this is repeated every 6 seconds. When the KLS is running, this operation increases greatly the amount of traffic. Moreover, the delay introduced by the KLS also affects the Skype's TCP operation. In particular, the TCP retransmissions, out of order packets and duplicate ACKs are 8 fold higher when using the KLS.

**Facebook:** we observe a favourable behaviour of the KLS when running Facebook. The packets are scheduled according to the algorithm in an energy efficient manner. Even though Facebook also uses TCP, no extra retransmissions, out of order packets or duplicated ACKs are observed and the amount of traffic is similar with and without the KLS.

To sum up, our observations reveal that shaping applications' outgoing traffic independently from their intention/behaviour can result in a counterproductive operation of some applications (e.g., Skype), whereas for others it can work out of the box (e.g., Facebook). In any case, if we do not make traffic distinction we cannot exploit the energy saving potential for all applications. Background traffic differentiation becomes an important piece of the energy puzzle in order to fully achieve energy savings.

## V. Related works

We categorise studies aiming to reduce energy consumption of 3G periodic transfers and background traffic based on their approach. Several works [3], [4] employ Fast Dormancy (FD) of the 3GPP Release 8 standard to control the radio resource from the UE. FD allows the UE to move to PCH before the expiration of the inactivity timers. The concept of traffic backfilling [18] is proposed to opportunistically transmit background data during unused gaps of energy tails between interactive traffic. Bartendr [8] schedules bulk data transmissions in periods of good signal strength. These works are complementary to our work since an implementation combining them would lead to higher energy savings.

Könönen et al. [13] propose the alignment of timers of different applications to perform synchronised bursty transmissions with a determined periodicity (traffic batching). Intentional Networking [5] forces developers to provide semantic labels to differentiate the data transmissions. Both approaches require the modification of applications and count on the good practices of developers, whereas our work attempts at reducing energy consumption at kernel level interacting only with the traffic. Our experiments confirm that distinction of application packets is a prerequisite for using kernel level shaping, even though in some cases (e.g. Facebook or instant messaging) it might work out of the box.

TailEnder [2] defers applications' data transmissions based on user-specified deadlines, leading to traffic aggregation and bursty transmissions while reducing the number of state transitions and energy. Traffic aggregation in their case defers within a maximum period the data transmissions until one of them needs to be sent (user-specified "deadline"), sending together all the transfers that were postponed until that moment. Calder et al. [12] schedule data transmissions of applications with different transmission intervals in multiples of the shortest interval. TailTheft [7] combines data aggregation and FD. CLBB [1] refines the above works by scheduling data transfers considering both the inactivity timers that lead to the energy tails and the RLC data buffers in combination. Qian et al. [19] perform a simulation comparison of different approaches (FD, traffic aggregation and batching) showing that a combination of approaches can lead to higher savings. A recent work by Huang et. al [11] show the impact of the traffic generated when the screen of the device is off on the energy consumption of 3G and 4G, and propose to employ FD and batching to reduce its energy consumption. Nguyen et al. [6] employ also user context information. In comparison, our work is the first implementation of such an energy saving approach at kernel level in a commodity device allowing us to quantify both its actual energy savings and its cost in terms of own energy footprint. Furthermore, we show the complexity of shaping the live traffic of real applications and the consequences for these approaches.

Based on the energy-delay trade-off, the same concepts can be also applied to other technologies. For example, Palit et al. [20] employ packet aggregation applied to WiFi, where packets are aggregated at the medium access control layer in order to fill a full frame and allow the device to extend its doze time. However, they do not evaluate the energy cost of their mechanism nor study the implications for real traffic.

## VI. Conclusion and future work

Achieving energy proportionality in systems that were designed for peak performance is a complex task. The focus of our work is on extending the battery lifetime of mobile devices by reducing the high energy overhead of the background traffic in 3G, which is mostly created in low utilisation periods.

In this work, we show that applying a mechanism that employs a small amount of energy to save energy pays off. We implement an energy-efficient algorithm to reduce the energy consumption of background traffic with no QoS requirements on the Android platform, and measure its overhead using controlled real traffic. The results show the low energy overhead introduced by our implementation at kernel level leads to net energy savings between 7% and 58% on emulated traces, depending on the amount of traffic and the maximum allowed delay.

We extend our evaluation using real live traffic as an exploration, and discover that the mix of data and background traffic destroys the energy saving operation of our kernel module (intended only for background traffic). However, there is evidence that for some low-intensity applications this would work out of the box.

Our future work involves traffic type differentiation at different layers to maximally exploit the energy saving potential. Traffic differentiation without application cooperation is an interesting approach to investigate.

## Acknowledgment

This work was supported by the Swedish national graduate school in computer science (CUGS). The authors wish to thank the support of Ericsson AB, and in particular B-O Hertz, Pär Emanuelsson and Claes Alströmer for providing the 3G developer kit and facilitating the measurement gathering phase. We would also like to thank Urko Zurutuza for his useful advice.

## References

[1] E. J. Vergara and S. Nadjm-Tehrani, "Energy-aware Cross-layer Burst Buffering for Wireless Communication," in *Proceedings of the 3rd International Conference on Future Energy Systems: Where Energy, Computing and Communication Meet*, ser. e-Energy '12. ACM, 2012.

[2] N. Balasubramanian, A. Balasubramanian, and A. Venkataramani, "Energy Consumption in Mobile Phones: A Measurement Study and Implications for Network Applications," in *Proceedings of the 9th ACM SIGCOMM Conference on Internet Measurement Conference*, ser. IMC '09. ACM, 2009, pp. 280–293.

[3] F. Qian, Z. Wang, A. Gerber, Z. Mao, S. Sen, and O. Spatscheck, "Top: Tail optimization protocol for cellular radio resource allocation," in *Procceedings of the 18th IEEE International Conference on Network Protocols (ICNP), 2010* , October 2010, pp. 285 –294.

[4] P. Athivarapu, R. Bhagwan, S. Guha, V. Navda, R. Ramjee, D. Arora, V. Padmanabhan, and G. Varghese, "RadioJockey: Mining Program Execution to Optimize Cellular Radio Usage," in *Proceedings of the 18th Annual International Conference on Mobile Computing and Networking (MobiCom)*, August 2012.

[5] B. D. Higgins, A. Reda, T. Alperovich, J. Flinn, T. J. Giuli, B. Noble, and D. Watson, "Intentional Networking: Opportunistic Exploitation of Mobile Network Diversity," in *Proceedings of the 16th Annual International Conference on Mobile Computing and Networking*, ser. MobiCom '10. ACM, 2010, pp. 73–84.

[6] N. Nguyen, Y. Wang, X. Liu, R. Zheng, and Z. Han, "A Nonparametric Bayesian Approach for Opportunistic Data Transfer in Cellular Networks," in *Proceedings of the 7th International Conference of Wireless Algorithms, Systems, and Applications (WASA)*, ser. Lecture Notes in Computer Science, Springer, 2012, vol. 7405, pp. 88–99.

[7] H. Liu, Y. Zhang, and Y. Zhou, "TailTheft: Leveraging the Wasted Time for Saving Energy in Cellular Communications," in *Proceedings of the 6th International Workshop on MobiArch*, ser. MobiArch '11, ACM, 2011, pp. 31–36.

[8] 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," in *Proceedings of the 16th Annual International Conference on Mobile Computing and Networking (Mobicom)*, ACM, 2010, pp. 85–96.

[9] F. Qian, Z. Wang, A. Gerber, Z. Mao, S. Sen, and O. Spatscheck, "Profiling Resource Usage for Mobile Applications: A Cross-layer Approach," in *Proceedings of the 9th International Conference on Mobile Systems, Applications, and Services*, ser. MobiSys '11. ACM, 2011, pp. 321–334.

[10] M. El-Gendy, A. Bose, and K. Shin, "Evolution of the Internet QoS and Support for Soft Real-time Applications," *Proceedings of the IEEE*, vol. 91, no. 7, pp. 1086 – 1104, July 2003.

[11] J. Huang, F. Qian, Z. M. Mao, S. Sen, and O. Spatscheck, "Screen-off Traffic Characterization and Optimization in 3G/4G Networks," in *Proceedings of the 2012 ACM Conference on Internet Measurement Conference*, ser. IMC '12. ACM, 2012, pp. 357–364.

[12] M. Calder and M. Marina, "Batch Scheduling of Recurrent Applications for Energy Savings on Mobile Phones," in *7th Annual IEEE Communications Society Conference on Sensor Mesh and Ad Hoc Communications and Networks (SECON)*, June 2010, pp. 1–3.

[13] V. Könönen and P. Paakkonen, "Optimizing Power Consumption of Always-on Applications Based on Timer Alignment," in *Proceedings of the 3rd International Conference on Communication Systems and Networks (COMSNETS), 2011* , 2011, pp. 1–8.

[14] C.-Y. Huang, C.-M. Chen, S.-P. Yu, S.-Y. Hsu, and C.-H. Lin, "Accelerate In-line Packet Processing Using Fast Queue," in *Proceedings of IEEE TENCON 2010*, November 2010.

[15] B. Leslie, P. Chubb, N. Fitzroy-dale, S. Gtz, C. Gray, L. Macpherson, D. Potts, Y. Shen, K. Elphinstone, and G. Heiser, "User-level Device Drivers: Achieved Performance," *Journal of Computer Science and Technology*, vol. 20, Springer, 2005.

[16] E. J. Vergara and S. Nadjm-Tehrani, "EnergyBox: A Trace-driven Tool for Data Transmission Energy Consumption Studies," in *Proceedings of the International Conference on Energy Efficiency in Large Scale Distributed Systems (EE-LSDS 2013)*, ser. Lecture Notes in Computer Science. Springer, April 2013.

[17] J. Sanjuan, "3G Energy-efficient Packet Handling Kernel Module for Android," (Student paper), Linköping University, 2012. [Online]. Available: http://urn.kb.se/resolve?urn=urn:nbn:se:liu:diva-84507

[18] H. Lagar-cavilla, K. Joshi, A. Varshavsky, J. Bickford, and D. Parra, "Traffic Backfilling: Subsidizing Lunch for Delay-Tolerant Applications in UMTS Networks," in *Proceedings of the 3rd ACM SOSP Workshop on Networking, Systems, and Applications on Mobile Handhelds, MobiHeld '11*, ACM, 2011.

[19] F. Qian, Z. Wang, Y. Gao, J. Huang, A. Gerber, Z. Mao, S. Sen, and O. Spatscheck, "Periodic Transfers in Mobile Applications: Network-wide Origin, Impact, and Optimization," in *Proceedings of the 21st International Conference on World Wide Web*, ser. WWW '12. ACM, 2012, pp. 51–60.

[20] R. Palit, K. Naik, and A. Singh, "Impact of Packet Aggregation on Energy Consumption in Smartphones," in *Proceedings of the 7th International Wireless Communications and Mobile Computing Conference (IWCMC)*, July 2011, pp. 589–594.