Piggyback Notification : Energy Efficient and Activity Aware Notification Scheduling

Utku Günay Acer, Afra Mashhadi, Claudio Forelivesi, Fahim Kawsar
Bell Labs, Alcatel Lucent

ABSTRACT

Mobile applications use the push notifications to inform users of updates and interactions. These notifications are pushed to the user devices by notification servers such as Apple Push Notification server) as they arrive from the third party content provider. However, due to their intrinsic small size, the push transfer is not power efficient, especially on cellular networks. Cellular network interfaces in mobile phones do not immediately go into low power idle-state after completing a data session. Instead, they remain in the high-power state for a certain duration, which results in a not utilized transmission channel and the higher energy consumption.

In this paper, we propose a scheduling technique for transmission of these small sporadic messages in an energy-efficient way. Our method sits on the cellular network side, thus does not require any modifications to the devices. It delays and piggybacks notifications to the future data transmission when possible. To achieve this, we employ an algorithm that based on the past user traffic, predicts the likelihood of the user's next activity in a given time and uses this information to schedule the notifications. Our findings show that by adopting such scheduling strategy, we are able to reduce energy consumption by 15%.

Categories and Subject Descriptors

H.4 [**Information Systems Applications**]: Communications Applications

Keywords

Push Notification, Network Sensing, Activity Prediction

1. INTRODUCTION

The push notifications have become increasingly prevalent and part of many smartphones applications, allowing the third party content providers to initiate communication with the users even when the application is not actively being used. This type of communication provides a visual or audible cue to inform the mobile users about new unattended messages or events. It has become so popular with some users receiving hundreds of push notifications per day [13, 15]. While the impact of this increase has been studied in terms of its disruptive aspects on the end-user, little attention has been given to the push notification delivery services from systems perspective and their impact on energy consumption on the mobile device.

The push notification messages are often small in size and sporadic in nature. Once issued by the third party content provider, these messages are sent to the push notification servers, such as Apple Push Notification service (APNs) or Google Cloud Messaging (GCM) which then transfers this information to the user device. The current implementation of push notification services is based on store-and-forward mechanism, pushing any notifications to the device as soon as the device is reachable. However, these services do not account for the devices' network interfaces, that is, after sending a message the cellular network interfaces in mobile phones do not immediately go into low power idle-state. Instead, they remain in the high-power state for a certain duration, often referred to as *tail time*. Therefore, delivering of small messages instantly and in isolation, as currently is done for push-notification messages, is an inefficient method which consumes unnecessary energy on the mobile devices. The amount energy spent during the tail time is called *tail energy*

A number of empirical studies have previously reported the effect of the tail time on the overall energy consumption of the mobile phones [12, 5]. In [5], authors have shown that the tail energy corresponds to the 60% of the total energy consumed by the device radio. These works propose systems to better utilize the tail time and reduce the tail energy. Common to all these works is that the system runs on the mobile devices, thus requiring changes to the device operating system and imposing a restriction on the scheduler, that is only the outgoing messages can be intercepted.

In contrast, we approach this problem from the network's perspective and propose a scheduling system for the push notification messages. Our method in the nutshell delays push notification messages when possible and sends them in batches or piggybacked with data traffic in an effort to conserve the energy spent as part of the tail time. More precisely, we design and develop a system which rather resides on the network operator hub and is capable of utilizing information regarding users past and current online activities (i.e., data traffic) to build a prediction model of users behav-

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.
Copyright 20XX ACM X-XXXXX-XX-X/XX/XX ...$15.00.

ioral patterns. We propose an algorithm that leverages this historical information to estimate the next upcoming activity of users and assigns a time-to-live (*ttl*) to the notification which best corresponds to the users temporal activity pattern. Our scheduling system then uses the *ttl* information to delay the notifications improving the energy consumption on the mobile devices. Indeed, our results based on cellular traces of 60 users suggest that using the scheduling approach we can decrease the energy consumption of mobile devices by 15% for the trade off of 150 seconds delay on average.

The remaining of this paper is structured as follows: we first provide background context on the cellular network interface energy consumption and the existing notification servers architecture in Section 2. We then describe the envisaged system and its components in Section 3. We describe the dataset at hand in Section 4.1, before presenting the results of our evaluation in Section 4. In Section 5, we present the current state-of-the-art that addresses the tail inefficiency. Finally we conclude the paper by discussing the limitations and implications of the proposed system in Section 6.

2. BACKGROUND

In this section, we briefly cover the background information regarding i) how resource allocation works on UMTS (Universal Mobile Telecommunication System) networks, and its implications on mobile device energy usage, ii) the architecture explaining how the push notifications currently work and are delivered to users devices.

2.1 UMTS and its Energy Model

The UMTS[a] network consists of three interacting elements: Core Network (CN), UMTS Terrestrial Radio Access Network (UTRAN), and User Equipment (UE). The UTRAN provides the air interface access method for User Equipment to connect to CN. It consists of two components Node-B (i.e., base stations), and Radio Network Controllers (RNC). RNC is a key element in the UMTS network and is responsible for the radio resource management as well as management of the multiple Node B instances (i.e., Base Stations), to which the UE connects through radio physical channel. The CN is the backbone of the cellular network and its main functions are to provide switching, routing and transit for user traffic, and to host user database and network management functions. The CN typically performs these operations via its Gateway GPRS support node (GGSN), responsible for the inter networking between the UMTS network and external packet switched networks, like the Internet and X.25 networks and Serving GPRS support node (SGSN), responsible for the delivery of data packets from and to the mobile stations within its geographical service area.

The power consumption of the UE is influenced by the Radio Resource Control (RRC) states and the Radio Link Control (RLC) protocols. A single RRC state machine is maintained at both the UE and the RNC (i.e., they are always synchronized), and its purpose is to *effectively utilize limited radio resources to improve power consumption.*

Typically there are three RRC states: IDLE, CELL_DCH, and CELL_FACH as shown in Figure 1. Each of these states are allocated varying radio resources and power.

[a]Although we focus 3G technology for the work in context, the basic principles of networking remain the same for 4G/LTE.

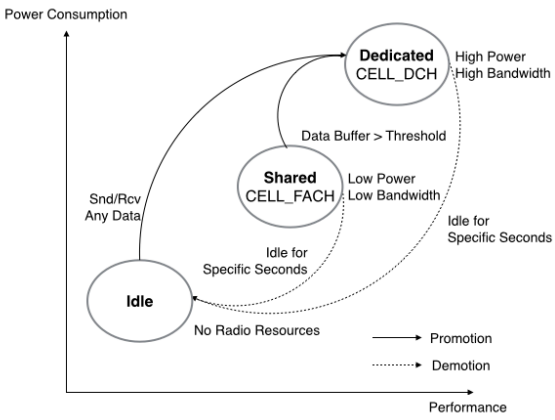

Figure 1: RRC State Machine for UMTS Network

- The IDLE state happens when there is no network activity on the device, e.g., when a handset is turned on and the RRC connection with RNC is not established. The power consumption of the radio interface in this state is almost zero. Any application traffic triggers a RRC state transition from IDLE state to the CELL_DCH.
- In the CELL_DCH, a RRC connection is established, and a *dedicated* physical channel is allocated for the UE in both uplink and downlink providing higher data rates.
- In the Forward Access Channel (CELL_FACH) the UE is assigned a default common or shared transport channel in the uplink and monitors the downlink continuously. The UE therefore can transmit small data packets at lower data rates while at CELL_FACH.

The RRC State transitions between these states are caused by traffic volume and inactivity timers controlled by the RNC. Most of the operators maintain statically set inactivity timers to control the state transitions from CELL_DCH to CELL_FACH as well as from CELL_FACH to IDLE. For example, when a UE is in the CELL_DCH state for a specific time period (which is called *tail time*) without any or small data transmission, the RNC releases the dedicated channel and switches the UE state to CELL_FACH. During the tail time, the UE waits for the inactivity timer to expire while maintaining transmission channels and its radio power consumption is kept at the corresponding level of the state. As such due to this tail time, transmitting even a small amount of data can cause significant radio resource and power consumption. This RRC State transitions are as the result particularly inefficient for small sporadic traffic such as push notifications.

2.2 Mobile Push Notification Service

Mobile push notification describes a style of internet-based communication where cloud-based applications can send brief alerts and updates to a client application running on a mobile device. The service provides a simple, lightweight mechanism to tell mobile applications to contact the server directly, to fetch the updated application or user data. Mobile operating system providers facilitate this service through

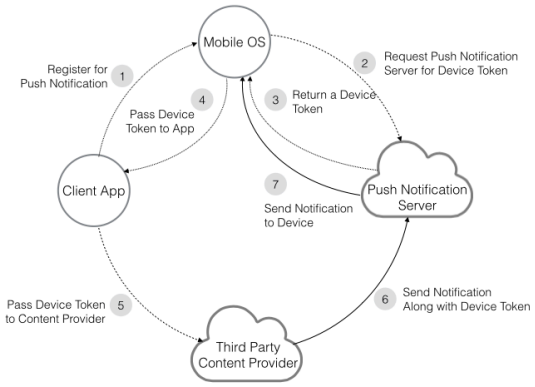

Figure 2: Architecture of Push Notification Service

dedicated notification servers, e.g., Apple Push Notification Server (APNs), Google Cloud Messaging Server (GCMs), etc. They are also responsible for all aspects of queuing of messages and delivery to the target application running on the target device.

Figure 2 illustrates interactions in the push notification eco-system where the stages are labelled numerically. The process starts with a client application requesting for a device token from the mobile operating system (1), which in turn contacts the respective notification server (2). This device token is then passed to the device operating system (3) and through the client application (4) to the content provider (5), which uses it for pushing subsequent notifications (6). Finally, the notification servers send this notification to the target device (7) by taking into account the notification type, its priority, expiry time and the availability of the device. While stage 1-6 describe an initialization and security handshake between the third content provide and the application, we are interested in stage 7 at which the delivery of the notification to the device occurs. This is done by the notification servers maintaining a persistent connection with the mobile devices. The typical payload of notification messages are fairly small. In addition, recent notification protocols also provide service providers with options for setting the expiry time and priority of a notification. For example, with APNs a notification message may include an expiration date that identifies when the notification is no longer valid and can be discarded from the APNs queue. If this value is non-zero, APNs stores the notification and tries to deliver the notification at least once within the stipulated time otherwise APNs delivers the notification immediately should the target device be reachable. Furthermore, a notification might include a priority flag that could be set by the content provider to indicate whether or not to delay the notification when the device is idle. For example, Microsoft Push Notifications(MPN) services allow for the content provider to append the NotificationClass in the HTTP header of the notification URI. The notification classes for Windows 8 are defined and fixed to i) Priority indicating the notification is delivered within 450 seconds; ii) Regular indicating the notification is delivered within 900 seconds; and iii) Real time indicating immediate delivery. These different classes allow the mobile OS and the MPNs to work in conjunction to batch notifications so as to save energy. Although, the current extensions allow for some level of control over the delivery of the push notifications, the delaying period defined by the push notification servers (e.g., MPNs) are arbitrary and not reflective of the device's RRC state.

2.3 Device-Centric Notification Management

Notification management on the device-end can be done either through applications or the OS itself. From the OS aspect, iOS, Android and Windows OS all provide the user with settings that allow them to indicate the willingness to receive notifications from their installed applications. These settings are limited to true/false with no personalization regarding the sensitivity and priority of the notifications from different applications. With regards to applications, various notification management applications are designed for Android OS. They mostly allow the user to classify notifications and organize them based on their preferences. While most of these applications provide an alternative interface to the notification center on Android devices, some also enable the user to ignore notifications from unwanted contacts or applications. However, any filtering on the notification is done at the device-end and requires the notification to be delivered to the device at the first place.

Alternatively, recipe based approaches, such as IFTTT (IF THIS THEN THAT), allow the user to set specific rules regarding their content from various channels (e.g., Facebook etc.) and the actions that are to be applied once a condition is met. Using this mechanism the user can create a more personalized notification delivery (e.g., when tagged in Facebook), enabling the IFTTT to pull content when the specified rule occurs. The IFTTT application persists a connection with the content provider channel through a polling period. However, in order to work as notification delivery, the polling period would need to be set to a very short time period (almost continuously). Thus making such application based approaches even more energy consuming than the traditional push notification delivery.

2.4 Push Notification Scheduler: A Remedy

Although, the notification servers offer some controls to content providers to qualify a notification in terms of its priority and delay tolerable property, they are not aware of the radio states of the target devices. Hence, high frequency of the notifications could increase the time during which the device ratio spends in tail time needlessly and thus draining significant amount of battery. In addition, the sporadic nature of the notifications also contributes to significant energy drain. This is because the majority of the network operators' dormant timer[b] is set to 30 seconds or less [2], thus making the delivery of sporadic notifications even more energy consuming. As Balasubramanian et al. have shown in [5], 60% of the total energy consumed for data communication corresponds to the tail energy, and the high frequency and sporadic bursts of notifications only adds to this drain. This is a remarkable overhead considering the past research has shown not all the notifications are perceived as important to the user and many of the notifications are ignored [13].

The work presented in this paper primarily aims to address this problem by delaying the notifications and piggybacking them with larger data traffic when possible. To accomplish this, our system is placed at the mobile operator

[b]The time based on which RNC releases high power state channels

end for accurate awareness of the radio states of the devices, and leverages a network traffic predicator to determine the appropriate delay interval for the notifications.

Such a system is made possible with the advent of the Network Function Virtualization (NFV), where the network components are no longer provided in closed "black-box" but are placed in virtualized servers [1]. This paradigm facilitates the network operators to offer new services to their users at little cost without complex modifications in the network components and any concerns about scalability of the component. Similarly in this paper, we exploit such underlying functionalities of NFV to design a push notification scheduler. In the next section, we present this system and discuss its constituent elements elaborately.

3. SYSTEM DESCRIPTION

In this section, we discuss the basic working principles of the proposed system for our envisioned notification scheduler. We propose to extend the basic functionality of the core network component of UMTS network by adding a component that is capable of delaying notification messages when possible to optimize the overall power consumption of the mobile device. We place this component on the data plane of the cellular core network instead of the RNC. By doing so, the module can infer when the device has any incoming or outgoing traffic, essentially making the push notification service aware of the radio state of device even though the actual state machine interface is not maintained there. It can be co-located with either the Serving GPRS Support Node (SGSN) and Gateway GPRS Support Node (GGSN) as these are the components that have functions on the data plane. The system is designed as a module composed of four elements, which leverage information from Deep Packet Inspection (DPI) capabilities:

1. *Burst Detector :* This element is responsible for capturing a group of traffic transactions (upload or download) that constitute a single burst of network activity.

2. *Notification Detector :* This elements is responsible detecting the push notification messages through TCP port monitoring and by leveraging the DPI module.

3. *Prediction Model :* This element uses the traffic burst pattern to maintain a model of a user's network activity, based on which it can provide an estimated delay that a notification can tolerate for that specific user.

4. *Notification Scheduler :* This element leverages the prediction model to determine the delay interval for the notification and accordingly schedule the delivery of the notification.

3.1 Burst Detector

A traffic burst represents a group of transactions under one single data transmission session. Typically a User Data Record (UDR) entry for a data transfer contains user identity as International Mobile Subscriber Identity (IMSI), transaction time, the amount of data being exchanged in the transaction, Node-B information, host information, etc. To determine a network activity session, it is necessary to identify and group a set of UDR entries together that appear in close temporal proximity and are separated by a specific time interval from the subsequent group of UDR entries. These groups of UDR entries or *traffic bursts* then can be ordered by time to construct a network activity trajectory, which can later be used for modeling temporal behavior of an individual. The Burst Detector works on top of the DPI module, and maintains a storage for each individual IMSI. As UDR entries come in for each IMSI, the Burst Detector checks the timestamp of the latest entry and compares it with the most recent entry to determine the interval. If the interval is longer than a selected threshold, then the set of entries that are in the storage are grouped as one burst and is passed to the Prediction Model as an identified traffic burst. Otherwise the entry is added to the storage as a constituent of ongoing burst. For the dataset presented in this paper and discussed in section 4.1, the interval threshold is set to 31 seconds which was selected empirically to minimize isolated transactions to appear as one small burst.

3.2 Notification Detector

The push notifications on mobile device typically use a particular port number while connecting to the push notification servers. For example, Android devices use TCP port 5228 to connect to GCM Servers. iOS devices on the other hand use TCP port 5223 to connect to the APN servers. Though the devices fall back to other ports in case they cannot establish connection on these ports (e.g., TCP port 443 port Android), it is safe to assume these ports are open on a cellular network. Hence, notifications can be detected by looking at the protocol field in the IP header to check whether TCP is used and which destination port is included in the TCP header.

3.3 Prediction Model for TTL Assignment

In order to ensure that our system accounts for users individuality and their different behavioral patterns, we require an adaptive mechanism which would assign a time-to-live (*ttl*) to each notification by evaluating its priority to the end user. It is possible to model this priority in two different ways. A *content-centric* approach in which the priority is decided based on the nature of the content and its sensitivity to delay. For example, a location sharing application may not tolerate any delay in its notification and thus require the *ttl* to be set to zero. It is worth noting that the Notification Servers such as APNs and GCM already account for this simple feature by allowing the application developers to set a priority field. Alternatively the priority can be set based on the popularity of the content and based on how it would be perceived by the user. A *user-centric* approach on the other hand accounts for the sensitivity of the user to the delay at a given time. In other words, the likelihood that the user would indeed attend the notification. Mashhadi et al. [13] has previously shown that the likelihood of a user attending a notification is much higher when the user is already engage with their phone. Similarly we take into account, user's engagement in an online activity on their device to model the delivery time of the notifications (i.e., *ttl*).

We design an algorithm that leverages the temporal activity[c] behavior of the user to estimate the likelihood of the user interacting with their mobile device at a given time. Using this adaptive mechanism, we are able to predict the *earliest* time that the user will start a data session and thus

[c]By activity we refer to users activity on the device which generates data traffic

assign the *ttl* accordingly. Although we grant a user-centric approach for our scheduling approach, it is worth noting that the two approaches of content and user-centric are not exclusive and could be combined in the future systems.

To predict the time of the upcoming network activity (i.e., next traffic burst) of a user, we build a $T \times N_l$ matrix U, where T denotes the number of time slots in a day and N_l is the number of look up days. Each element u_{ij} in U is a binary that represents whether a user's was engaged in any network activity at the time slot i of the day j ($i \in T$ and $j \in N_l$). Our algorithm uses U to predict whether a network activity is likely to occur in the next prediction horizon window (denoted as f) of the current day by matching patterns of similar days in the past N_l days. The current day and the current time slot are denoted as j_c and i_c, respectively (i.e., the current network activity value is $u_{i_c j_c}$). As a day progresses, network activity vector is constructed from midnight up to the current time. To predict the next upcoming networking activity upon the arrival of a notification at time $t_{notf} = i_c$, we set a look up window of k immediate past slots denoted as L_k and run a similarity measure that compares the network activity of past L_k slots of the current time/day (i_c) with the corresponding time slots of previous N_l days. M top most similar days are selected where $M < N_l$.

To obtain a similarity value of the current day to the past days we compute a binary similarity measure [6]. We have examined several binary similarity measures with our dataset by dividing our samples into subsets randomly. We have found that Sokal-Michener measure offers the best discrimination capability [19]. Therefore, to compare the similarity between two temporal activity vectors x and y with length k representing activity bits of past L_k slots of today and one of the past N_l days respectively, we define a *temporal day similarity score* for each of the j^{th} day as:

$$DS_j = \frac{1}{k} \sum_{s=1}^{k} I(x_s^t y_s || \overline{x}_s^t \overline{y}_s), \tag{1}$$

where $I(r)$ is the indicator function, and $I(r) = 1$ if r is true or 0 otherwise, $x_s^t y_s$ denotes the positive match and $\overline{x}_s^t \overline{y}_s$ denotes the negative match at k^{th} position between x and y,

Once the day similarity scores are obtained, the algorithm moves to the selection of the M candidate days. This selection is performed by sorting N_l past days twice, first on the day similarity score and sorting on the time difference from the current day. It is worth noting that in cases where there are no similar days ($DS_j = 0$), the algorithm will pick the candidate days based on the most recent ones. Finally, the prediction algorithm considers the network activity vector of each candidate day for the upcoming slots (prediction horizon window f) and computes the probability of occurrence of a network activity, e.g., presence of network traffic for each of the upcoming slots. If the probability is higher than a selection threshold p_{th} then that network activity in the prediction slot is set to 1 or 0 otherwise. After ROC analysis, we set p_{th} to 0.6. The algorithm performs this step by taking each slot at a time and combining day similarity score to ensure that most similar and more recent days have highest contribution in predicting the occurrence of a network activity. The *ttl* is then set to the first time slot which is predicted to have an activity ($u_{i_f j_c} = 1$).

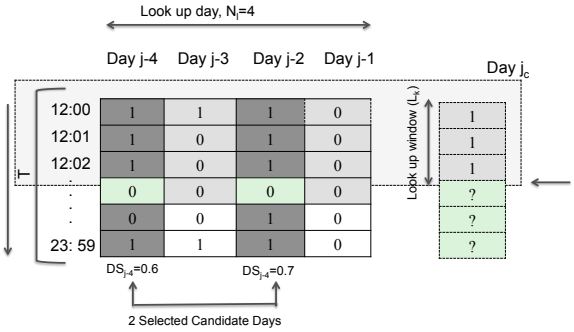

Figure 3: Example of the Network Activity Prediction and TTL Estimation Process

$$u_{i_f j_c} = \begin{cases} 1 & \text{if } \left(\sum_{j=1}^{M} DS_j\right)^{-1} \sum_{j=1}^{M} DS_j I(u_{i_f j} = 1) > p_{th} \\ 0 & \text{otherwise} \end{cases} \tag{2}$$

When the prediction slot i_f is at the beginning or end of a day, i.e. before midnight or just after midnight, the lookup slots for similarity matching are determined from the immediate previous day, and candidate slots for prediction are selected from the immediate next day. This avoids complexities in making predictions that span midnight. Algorithm 1 summarizes the described prediction method.

Algorithm 1: The Activity Prediction Algorithm

Input: Lookup Days N_l, Current Day j_c, Current time slot i_c, Number of Candidate Days M, Lookup Window L_k
Output: ttl

```
Initialize the Matrix U and its element corresponding to temporal activity vectors u_ij, i ∈ [0,T] for N_l days
Initialize a Column Matrix U_c for current day j_c, and its temporal element u_ij_c, i ∈ [0, i_c]
for j = 0 to Lookup Days (N_l − 1) do
    Construct lookup vector for j^th day
    Compute the day similarity score DS_j using Equation 1
end
Sort N_l Lookup Days first based on day similarity and then on Time Difference from current day and select top M Days; i_f = i_c;
while j == j_c do
    Compute the activity occurrence probability at prediction time i_f using Equation 2
    if p_{u_{i_f j_c}} > p_th then
        u_{i_f j_c} ← 1;
        ttl = i_f − i_c;
        return ttl
    else
        u_{i_f j_c} ← 0;
        Continue;
    end
end
```

Figure 3 illustrates a simplified example, where the number of look up days N_l are set to 4 and the temporal granularity is in minutes that is $T = 1440$ (matrix U is 1440×4). In this example a notification has arrived at time $t_{notf} =$

12 : 03, and we want to estimate the next upcoming slot i where there would be any networking activity. The algorithm starts with detecting the similar past days to the current day j_c from the look up window N_l, where 2 days are selected. We then compute by considering the candidate days ($j-4$ and $j-2$) the likelihood for the activity at i_c+2 would be 1 with $p_{th} = 0.6$. The algorithm stops when it finds a candidate time slot (i.e., of predicted value 1) and would return *ttl* as 2 minutes, i.e., 120 seconds.

3.4 Notification Scheduler

The notification scheduler has the ability to detect if a packet is a notification and can intercept traffic. Traffic that corresponds to the data messages rather than isolated notification messages are sent immediately to the users. The notification messages on the other hand are delayed to conserve energy on the devices. The decisions to delay the notifications are governed by the three following heuristics.

3.4.1 *Heuristic 1:* Send Immediately

With this mechanism, no scheduling is performed. This simple approach mimics the state-of-the-art notification delivery in the network that immediately sends the notification to the destination device.

3.4.2 *Heuristic 2:* Send with Next Non Notification Data

In this heuristic, a notification message for a device is suspended in the notification scheduler in the core network until a data message arrives for or from the device. Upon the arrival of the data message, the notification messages are batched and piggybacked to the data message. This heuristic corresponds to a naive scheduling where all the notifications are delayed without any prediction of user activity.

3.4.3 *Heuristic 3:* Send After TTL Expiration or with Next Non Notification Data

In this heuristic, we introduce the *ttl* into scheduling. Recall from the previous section that each notification is associated with a *ttl* by which the messages needs to be sent. In accounting for *ttl* we add an expiration timer to the queue that holds all the notification messages destined for a device, that is set to the minimum *ttl* among the messages in the queue. The scheduler sends all the notifications to the destination when the timer expires. If a data message that can not be delayed arrives before the timer expires, the scheduler piggybacks the notification messages to the data and cancels the existing timer.

4. EVALUATION

In this section, we first discuss the dataset used in our analysis. We then discuss the performance of the prediction algorithm and the notification scheduler.

4.1 Datasets

We obtained a dataset of anonymized network activities of 60 users for over a month period in March 2013.[d] This data comprises the *size*, *time* and the *duration* of all incoming and outgoing network traffic for each user. In order to preserve the privacy of the users, the users are presented by random ids and their phone number is dropped from the database. Additionally any information regarding the location (e.g. the cell tower location, GPS etc.) has been also removed from the dataset. The data at hand is restricted in two aspects, firstly it does not contain any host URL information, making it impossible to determine whether a packet was sent by a push notification server; secondly it does not include the destination port number, thus making it further difficult to detect push notifications by monitoring the traffic to the specific port (e.g., port 5028 for Android devices and 5223 for iOS).

Therefore, in order to identify the push notifications for the purpose of testing our scheduling algorithm, we require an alternative way for identifying the notification messages. To do so, we assume that the download traffic packets which are *small* in size and has occurred in *isolation*, that is they are neither triggered or followed by any upload requests from the device, correspond the push notifications. In other words, the traffic that has occurred at least t_{tail} seconds after its preceding traffic and is not followed by any other traffic for at least t_{tail} seconds; where t_{tail} is the tail time parameter. We set the size for the determined notifications to less than 300 bytes as it corresponds to the iOS notifications defined by APNs (256 bytes) at the time data is collected. We set t_{tail} to 30 seconds as it is commonly used by the network operators.

Although we cannot ensure that all the detected small isolated traffic from our dataset are indeed the push notification messages (we may have false positive), we can argue that all the push notifications in our dataset are detected (no true negative). In other words, alongside those detected push notifications we may also observe other packets that are small isolated download traffic such as those packets corresponding to advertisements. We argue that should these false positive cases happen to be part of a larger traffic, they would be have been followed by immediate data transaction either before or after them. However, as they are detected in isolation, they correspond to the cases where the data transmission would have caused the cellular interface to stay unnecessary in the tail state with no upcoming traffic.

The resulting dataset consists of 143,261 isolated notifications and 765,739 traffic bursts. Figure 4 further illustrates the number of detected notifications for each user per day. On average we find that the users received around 80 notifications per day (median=74). However, as seen from Figure 4, some users receive as many as 200 notifications on some days and only a few on other days. This trend corresponds in size and distribution to those push notifications collected by Pielot et al [15] and also those observed by Mashhadi et al. [13] from empirical study of notifications on the mobile devices.

4.2 Performance of Prediction Algorithm

Predicting an activity pattern for a future hour slot is essentially a multi-label classification problem and the performance of the algorithm can be evaluated by standard Information Retrieval measures for a multi-label classification setting. For each time slot i_f, let T be the true network activity, and S be the predicted set activity. Accuracy is measured by the Hamming Score which symmetrically measures how close T is to S, i.e., $Accuracy(i_f) = \frac{\|T \cap S\|}{\|T \cup S\|}$. Precision ($P$), Recall ($R$) and F-Measure ($F_1$) are defined as

[d]Specific details about the identity of the network provider, as well as the location and time of the UDR data are omitted in order to preserve the anonymity of the network operator.

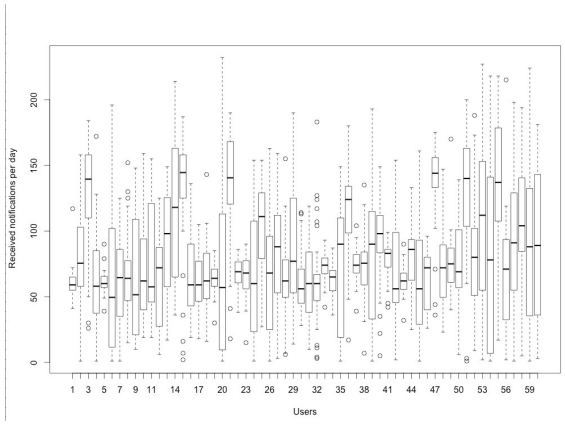

Figure 4: Number of notifications per user per day.

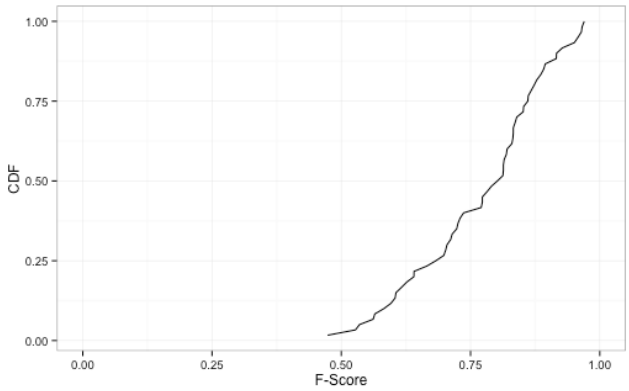

Figure 5: Cumulative Distribution of Prediction Performance over all 60 Users

$P(i_f) = \frac{\|T \cap S\|}{\|S\|}$, $R(i_f) = \frac{\|T \cap S\|}{\|T\|}$ and $F_1(i_f) = \frac{2P(i_f)R(i_f)}{P(i_f)+R(i_f)}$. For evaluating the algorithm, we split 30 days of data into two parts. The data of first 15 days are used to train the algorithm, e.g., as histories of network activities and the data of remaining 15 days are used to evaluate the performance of the algorithm.

Figure 5 plots the cumulative distribution of F-Score across all 60 users with 3 minutes as temporal slot duration, 3 hours as lookup slot duration, 15 look up days, 5 candidate days, and 0.6 as the selection threshold. As we observe, at least 60% of the users have over 0.7 F-Score, which is considered reasonably high. Prediction performance remains consistent with varying lookup and candidate days. Figure 6 illustrates the prediction performance and corresponding TTL assignments of the varying temporal slots (in seconds). We observe that larger temporal slots increase the prediction accuracy substantially, e.g., F-Score approaching to 1, as the users are certain to have network traffic on larger temporal interval. The reverse is also true, i.e., with smaller slot duration, e.g., 1 minute prediction performance falls to 0.4 for 50% of the users.

In the next subsection, we evaluate the performance of notification scheduler through a simulation model in which we construct the prediction model with the parameter as depicted in table 1.

Parameter	Value
Lookup Days	15
Candidate Days	5
Slot Duration	180 Seconds
Lookup Slot	3 Hours
Selection Threshold	0.6

Table 1: Prediction Parameter for the Simulation Model

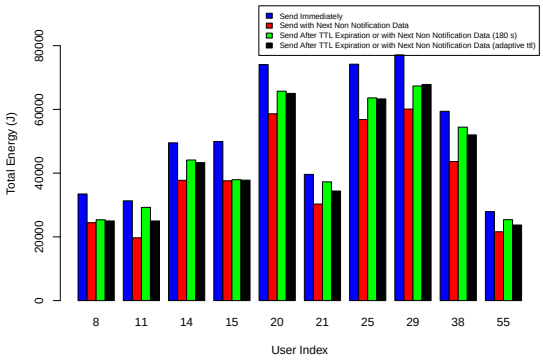

Figure 7: Energy consumption for each user with scheduling heuristics.

4.3 Performance of Notification Scheduler

In order to evaluate the performance of our scheduling system, we perform trace driven simulations using the dataset. We use a server-client model to simulate the core network. In this setup, there exists a number of clients and each corresponds to a base station. The client hosts a number of threads each modeling the radio state machine of a user. The state machine transitions are triggered by the arrival or transmission of data messages and the expiration of inactivity timers as explained in Section 2.1. The power consumption levels in each state and the timer expiration duration values are adopted from [10].

The downlink traffic is sent from the server to the client and the client pushes it to the module that corresponds to the user. The uplink traffic on the other hand is originated by the user module and pushed to the client and the server. The information regarding the data traffic is deducted from the real data traces of the same 60 users. The module calculates the energy consumption by the radio using the size of uplink and downlink traffic.

The server acts as the location where notification scheduler is placed that forwards the packets to the their destinations. Once a notification is detected, the decision to suspend is performed through the three described heuristics (Section 3.4).

Figure 7, illustrates the results of our energy saving by comparing the radio energy consumption for 10 selected users with all the heuristics. Even though, we only show 10 users for the clarity in the presentation, the results are consistent with the other users as well. The figure illustrates the inefficiency of the current notification delivery approach (i.e.,*Send immediately*). Looking deeper at the results we can observe that by suspending the delivery of the notifica-

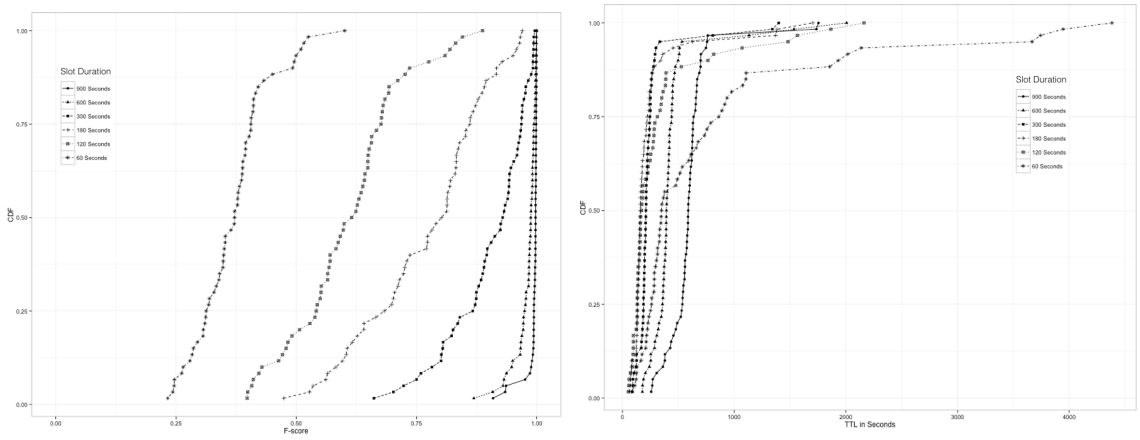

Figure 6: Influence of Varying Slot Duration (in seconds) on the Prediction Performance and TTL Estimation

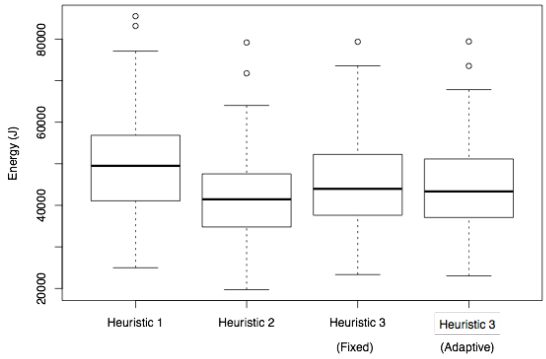

Figure 8: Combined energy consumption of all devices.

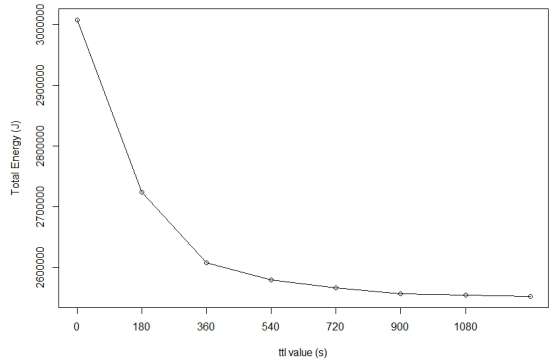

Figure 9: *ttl* value vs Energy. Energy consumption reduces as *Send After TTL Expiration or with Next Non Notification Data* uses larger fixed *ttl* values.

tion until the *Next non-notification data transfer*, we can on average conserve 15% energy (stdev=0.06), and that is up to 38% for the top user.

Next, we evaluate the *Send After TTL Expiration or with Next Non Notification Data* heuristic both with a fixed *ttl* and an adaptive *ttl* for each notification using the prediction algorithm. When the *ttl* is fixed, the scheduler sends the notifications by latest when the fixed *ttl* has reached. We evaluate by setting the fixed *ttl* value for this heuristic to 180 seconds. Comparing the two we can observe from the Figure 7 (and for the all the 60 users) that with the adaptive *ttl* the energy consumption is lower (μ = 44790.37 Joules, δ = 12798.86) than the fixed *ttl* (μ = 45411.57 Joules, δ = 12529.88), as the notifications are typically suspended for a longer duration.

Figure 8 shows the amount of energy consumption by all the users in a span of 1 month. Heuristics corresponds to those defined earlier with the addition of variant of Heuristic 3 with the fixed (*ttl*=180 secs) and adaptive. *Send with Next Non Notification Data* saves 16% energy in comparison to *Send Immediately*. The energy savings for *Send After TTL Expiration or with Next Non Notification Data* with adaptive *ttl* amounts to 11%. This heuristic saves as much as 10% energy even with a fixed *ttl* as low as 180 seconds.

In Figure 9, we show how the total energy consumption across all the users changes when *Send After TTL Expiration or with Next Non Notification Data* heuristic uses various fixed *ttl* values. Note that *ttl*=0 corresponds to *Send Immediately*, and the last data point corresponds to *Send with Next Non Notification Data* heuristic, which gives the lower bound in energy consumption. We see that as the *ttl* increases, the resulting energy consumption approaches the lower bound. When the *ttl*=18 minutes (1080 seconds), the energy consumption is only about 1% more than the lower bound.

Figure 10 shows the latency caused by the notification scheduling mechanisms for the set of selected 10 users. As before, the results are consistent with all the users even though exact values may change. From this figure we observe that *Send Immediately* heuristic immediately sends the notifications without intercepting them, it does not cause any latency. *Send with Next Non Notification Data* suspends the notifications for a longer duration than the other heuristics. These findings are parallel with the energy ob-

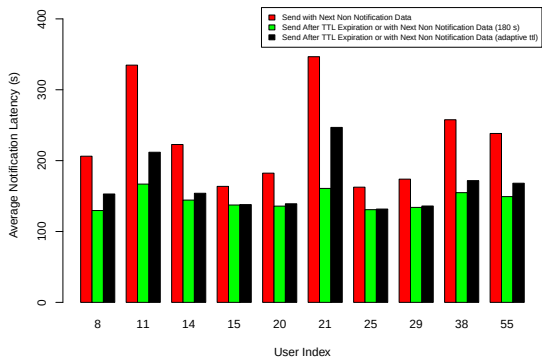

Figure 10: Latency of the notification delivery for each user with various heuristics.

Heuristic	**Average Latency (in secs)**	**Average Energy Saving**
Send with Next Non Notification Data	230	15%
Send After TTL Expiration or with Next Non Notification Data (ttl = 180 s)	136	9%
Send After TTL Expiration or with Next Non Notification Data (adaptive ttl)	157	10%

Table 2: Latency versus Energy Saving for the Scheduling Heuristics

servations in Figure 7. To present this trade-off the overall latency introduced by the heuristics is reported in Table 2 along with the average percentage of energy *saving*. These results are encouraging as they confirm the performance of our scheduler. That is although *Send After TTL Expiration or with Next Non Notification* with adaptive ttl consumes only 5% more energy than the lower bound in *Send with Next Non Notification Data*, it takes 32% less time to deliver notifications.

4.4 Limitation

In this paper we have designed a system for scheduling the notifications and where possible we have adhered to a generic solution. However, due to the limitations imposed by the data at hand, we had to apply some restrictions to the evaluation of the proposed system.

We granted a trace-driven analysis in order to evaluate the energy savings on the end-user devices. We understand that as the result of this approach, we are indeed facing a "Butterfly Effect" where tempering with an initial condition (i.e., delaying a notification) may result in a different course of future events (e.g., delayed communication) than what we originally have in the traces. Although, the course of future data traffic that is exhibited may be different, we believe our main findings regarding the quantification of the energy saving stays valid. The values reported in this section are not meant to be taken as absolute values but rather as bounds in the extent of energy savings.

In selecting the required parameters for our system, we had to adhere to an assumption that all the notifications are issued by APNs and are intended for iOS devices, that is they are less than 300bytes in size. We adhered to this choice as our data does not include information regarding the individual devices or port numbers. Furthermore, the alternative thresholding for Android notifications is indeed much bigger in size which would have result in the detection of many more notifications and a better performance of our system.

5. RELATED WORK

In [5], authors show that the energy consumed in the tail state corresponds to the 60% of the total energy consumed for data communications. They propose an algorithm that schedules and aggregates the outgoing small transmissions into large ones so that the occurrence of tails (and thus energy consumption) can be reduced. In [8], the authors use machine learning techniques to predict the network activity on the phone, which is used to manipulate the state transitions on the phone. Another relevant technique is TailTheft [12], which uses the duration of tail state to prefetch content and deferred messages for delay tolerant applications. Sending location update messages in an energy efficient way is addressed in [4]. In this work, messages to the location servers may be delayed as long as some quality measures such as time elapsed since the last message, the distance from the last reported location etc. are satisfied. In all these works, the traffic scheduling is considered only for outgoing packets as the scheduler runs on the phone, and requires modifications on the operating system or on the apps as well as resources to perform these computations. In contrast, our method requires no changes to the mobile phone and runs in the network.

A broad body of research addresses tail tuning and termination. In [7] and [9], different values for tail time has been suggested and proven to reduce the energy consumption. In [18, 17, 11] dynamic assignment of tail time based on usage pattern has been proposed. In particular [18] relies on prediction and dynamically terminates the tail time when it does not foresee an upcoming activity. Similarly [3] proposes data mining approaches to detect end of communication spurts to invoke fast dormancy with higher accuracy. Finally [16, 20, 14] focus on modeling and profiling the resource usage on mobile devices. In [16] the authors introduce ARO (Application Resource Optimization) tool that helps the developers to discover the inefficient resource usage by considering a cross-layer interaction for layers ranging from cellular interface to application layer.

6. DISCUSSION AND CONCLUSION

In this paper we have designed a system for scheduling the push notifications that resides in the core network of a cellular network and is added to the data plane. Our system leverages the paradigm of Network Function Virtualization (NFV). With this paradigm, network components are provided in software in virtualized servers and not in dedicated and specialized hardware platforms. In this way, our scheduler can scale up and down by simply initiating or shutting down virtual machines in which the scheduler runs. The proposed system suspends the notification messages in the network and leverages a DPI that monitors and exploits its

past activity to predict when she will be engaged with her device in the near future. This information is then used to schedule the queued notifications. Our results show that in a UMTS network setting, the power savings can be as high as 38% per user.

Although, we presented our approach for the UMTS setting due to the dataset at hand (extracted from UMTS network), our system can easily be adapted to the younger generation networks such as LTE. In a UMTS network, the scheduler can be situated along with Serving GPRS Support Node (SGSN) or Gateway GPRS Support Node (GGSN). In LTE evolved packet core (ePC), the scheduler may be situated with data plane elements Service Gateway (SGW) and the PDN gateway (PGW). SGW is a better option as the location of the scheduling module a mobile device may use more than one PGWs but is connected to a single SGW instance. This way, the scheduler has access to a user's all traffic activity. Due to the nature of DRX (Discontinuous Reception) mechanism which involves a state machine similar to the one used by UMTS, we believe the savings could be *more significant* with LTE networks. As the previous work has shown that the effect of the tail energy is even more drastic in LTE networks than 3G networks[10].

Another aspect of our implementation is that an instance of components like GGSN and SGW only serve a particular geographical area. We do not introduce any handoff mechanism for users moving from one area to another area. Rather, we assume the notification scheduler in an area only stores and uses data regarding user behavior in that geographical region. In so doing, we are implicitly introducing a spatial element to the notification scheduler. As the by product of this design decision, our activity prediction could be extended to take into account the spatial features (e.g., activity in significant places such as home) in addition to the current temporal ones. Since the scheduler is implemented in virtualized servers, the system does not need to concern with the inter radio access technology handovers either. The same servers can be utilized with UMTS and LTE network in areas where both services are available.

Moreover, our evaluation was performed on a dataset that extracted from a core network in early 2013. Since then, mobile operating system vendors allow for a larger payload in the notification messages sent by the application providers. In 2013, an APNs notification could fit in a single network packet. However, the payload size can now be as much as 4 Kbytes. With a typical MTU size of 1500 bytes, this can fit in at most four packets considering the lower layer networking overhead. This is still not a large stream and with the state-of-the-art optimizations in TCP, all packets are sent at once without waiting for the acknowledgement for the first packet. Therefore, our approach would work on the current implementation of push notification servers without a need to make any modifications to our system to address such payload sizes.

Finally, this work has important implications regarding conserving tail time energy on end-users device. As our findings have demonstrated, in many cases isolated notifications are pushed by the notification servers such as APNs causing an inefficient establishment of the cellular interface on the device. While the impact of this problem, might not be as exhaustive on today's smart-phones where the predominantly used lithium-ion batteries have sufficiently hight capacity, it is to a much greater extent on wearables and other devices where the energy resources are scarce due to form factor[e]. Although, this problem could also be addressed by including time-to-live and priority of the notifications at the content provider side while leveraging an activity prediction *from the device OS*, such approach would not be feasible for devices with limited processing powers. Therefore, we strongly believe the answer in addressing this challenge lies within the network's end as they have a bigger picture of users activities and devices' cellular interfaces, without the need to rely on the device to perform any calculations or communications.

7. REFERENCES

[1] Network Functions Virtualisation, An Introduction, Benefits, Enablers, Challenges & Call for Action. White Paper, SDN and OpenFlow World Congress, Oct 2012.

[2] G. Association. Fast dormancy best practices, version 1.0, 2011.

[3] P. K. Athivarapu, R. Bhagwan, S. Guha, V. Navda, R. Ramjee, D. Arora, V. N. Padmanabhan, and G. Varghese. Radiojockey: Mining program execution to optimize cellular radio usage. In *MobiCom*, 2012.

[4] P. Baier, F. Dürr, and K. Rothermel. Opportunistic position update protocols for mobile devices. In *Ubicomp*, 2013.

[5] N. Balasubramanian, A. Balasubramanian, and A. Venkataramani. Energy consumption in mobile phones: a measurement study and implications for network applications. In *IMC*, 2009.

[6] S. Choi, S. Cha, and C. C. Tappert. A Survey of Binary Similarity and Distance Measures. *Journal of Systemics, Cybernetics and Informatics*, 2010.

[7] M. Chuah, W. Luo, and X. Zhang. Impacts of inactivity timer values on umts system capacity. In *WCNC*, 2002.

[8] S. Deng and H. Balakrishnan. Traffic-aware techniques to reduce 3g/lte wireless energy consumption. In *CoNEXT*, 2012.

[9] H. Falaki, D. Lymberopoulos, R. Mahajan, S. Kandula, and D. Estrin. A first look at traffic on smartphones. In *IMC*, 2010.

[10] 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 *MobiSys*, 2012.

[11] C.-C. Lee, J.-H. Yeh, and J.-C. Chen. Impact of inactivity timer on energy consumption in wcdma and cdma2000. In *Wireless Telecommunications Symposium, 2004*, 2004.

[12] H. Liu, Y. Zhang, and Y. Zhou. Tailtheft: Leveraging the wasted time for saving energy in cellular communications. In *MobiArch*, 2011.

[13] A. Mashhadi, A. Mathur, and F. Kawsar. The myth of subtle notifications. In *Ubicomp*, 2014.

[14] A. Pathak, Y. C. Hu, and M. Zhang. Where is the energy spent inside my app?: fine grained energy accounting on smartphones with eprof. In *EuroSys*, 2012.

[15] M. Pielot, K. Church, and R. de Oliveira. An in-situ study of mobile phone notifications. In *MobileHCI*, 2014.

[16] F. Qian, Z. Wang, A. Gerber, Z. Mao, S. Sen, and O. Spatscheck. Profiling resource usage for mobile applications: A cross-layer approach. In *MobiSys*, 2011.

[17] F. Qian, Z. Wang, A. Gerber, Z. M. Mao, S. Sen, and O. Spatscheck. Characterizing radio resource allocation for 3g networks. In *IMC*, 2010.

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

[19] R. R. Sokal and C. D. Michener. A Statistical Method for Evaluating Systematic Relationships. *University of Kansas Scientific Bulletin*, 38:1409–1438, 1958.

[20] L. Zhang, B. Tiwana, Z. Qian, Z. Wang, R. P. Dick, Z. M. Mao, and L. Yang. Accurate online power estimation and automatic battery behavior based power model generation for smartphones. In *CODES*, 2010.

[e]http://www.newelectronics.co.uk/electronics-blogs/powering-wearables-and-giving-batteries-a-better-life/64664/