2014

SPS: an SMS-based Push Service for Energy Saving in Smartphone's Idle State

Erich Dondyk
University of Central Florida

SPS: AN SMS-BASED PUSH SERVICE FOR ENERGY SAVING IN SMARTPHONE'S IDLE STATE

by

ERICH DONDYK
B.S. University of Central Florida, 2012

A thesis submitted in partial fulfilment of the requirements
for the degree of Master of Science
in the Department of Electrical Engineering and Computer Science
in the College of Engineering and Computer Science
at the University of Central Florida

Spring Term
2014

Major Professor: Cliff Zou

ABSTRACT

Despite of all the advances in smartphone technology in recent years, smartphones still remain limited by their battery life. Unlike other power hungry components in the smartphone, the cellular data and Wi-Fi interfaces often continue to be used even while the phone is in the idle state to accommodate unnecessary data traffic produced by some applications. In addition, bad reception has been proven to greatly increase energy consumed by the radio, which happens quite often when smartphone users are inside buildings. In this paper, we present a Short message service Push based Service (SPS) to save unnecessary power consumption when smartphones are in idle state, especially in bad reception areas. First, SPS disables a smartphone's data interfaces whenever the phone is in idle state. Second, to preserve the real-time notification functionality required by some apps, such as new email arrivals and social media updates, when a notification is needed, a wakeup text message will be received by the phone, and then SPS enables the phone's data interfaces to connect to the corresponding server to retrieve notification data via the normal data network. Once the notification data has been retrieved, SPS will disable the data interfaces again if the phone is still in idle state. We have developed a complete prototype for Android smartphones. Our experiments show that SPS consumes less energy than the current approach. In areas with bad reception, the SPS prototype can double the battery life of a smartphone.

ACKNOWLEDGMENTS

Foremost, I would like to express my sincere gratitude to my advisor Dr. Cliff Zou for his guidance, consideration, and support throughout my graduate and undergraduate studies. Through his advising I gained insight and perspective that will remain with me throughout my entire career.

I would also like to thank my committee: Dr. Mainak Chatterjee and Dr. Kien A. Hua for their support and valuable insight.

TABLE OF CONTENTS

LIST OF FIGURES

CHAPTER 1: INTRODUCTION

Smartphones have penetrated the technology market at a staggering rate. In fact, it is estimated that currently more people own smartphones than personal computers throughout the world [1]. However, despite of the tremendous functionality smartphones provide, they still remain plagued by a short battery life.

One of the main sources of energy consumption in smartphones is the wireless radio. Specifically, the data network interfaces used for cellular data, such as 3G and 4G, and Wi-Fi. The data interfaces inherently need a significant amount of energy when transmitting and receiving data. In addition, tail power phenomenon introduces additional energy consumption for every retransmission. The cause and possible solutions to these internal sources of energy consumption have been explored in other works [2] [3] [4]. However, there are also two external factors that can significantly contribute to total amount of energy the data interfaces consume: bad reception and idle state data traffic.

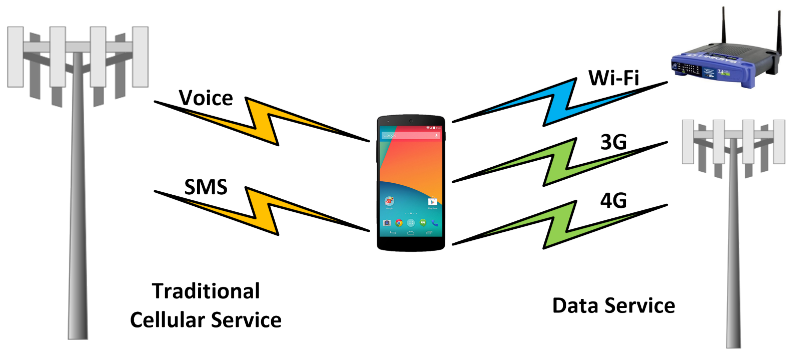


Figure 1.1: The communication interfaces of a smartphone. The data interfaces, such as 3G, 4G, and Wi-Fi, are among the components of the smartphone that consume the most energy. Our proposed energy-saving scheme exploits the traditional cellular short message service (SMS) to facilitate real-time data notification service that currently solely relies on the data service channels.

Bad reception introduces a series of side effects that can greatly increase the amount of energy that the data interfaces consume. First, smartphones experiencing bad reception frequently dissociate and reassociate with base stations or switch between different cellular technologies such as 3G and 4G. This frequent switching back and forth, known as the ping-pong effect, produces additional energy costs. Secondly, bad reception often causes retransmissions at the physical and transport layers which also produces additional energy costs. Finally, the smartphone radio is design to adjust their parameters when experiencing bad reception in order to increase throughput and decrease bit error. These adaptations generally result in lower data rates and higher transmission powers both of which consume more energy. With the average smartphone experiencing bad reception 47% of the time [5], the aggregate impact of these side effects can significantly reduce the battery life.

An average smartphone user interacts with his smartphone for only 58 minutes throughout a day [6]. The remaining of the day smartphones are usually in the idle state. While in the idle state, components such as the display and the processor are normally turned off or placed in low power states to conserve energy. However, unlike other components, the data interfaces often continue to be used during the idle state to accommodate the data traffic generated by some applications. In fact, study [5] shows that 19% of all the traffic generated by a smartphone is generated during the idle state. Most of this idle state traffic is unnecessary because it provides no immediate functionality to the user and it is often the result of careless application design.

On the other hand, some of the idle state data traffic is necessary. For example, new email arrivals, social status updates, and instant messages all require real-time notification to their users. Currently, push services are the mechanism by which notifications are delivered to smartphones, which rely on long-lived TCP connections between smartphones and the push server. Such a connection serves as a "virtual circuits" by which notification data that the smartphone has not request can be "pushed" to the smartphone. Hence, the data interfaces are required to remain active even during the idle state in order for the smartphone to maintain data channel to receive notifications.

In this paper, we present a Short message service Push based Service (SPS) to save unnecessary power consumption when smartphones are in idle state, especially in bad reception areas. SPS disables a smartphone's data interfaces whenever the phone is in idle state—this will remove all the energy wasted in maintaining data interfaces in bad reception areas in idle state, and remove energy consumed by the unnecessary idle state data traffic. To preserve the real-time notification functionality required by some apps, such as new email arrivals and social media updates, when a notification is needed, a wakeup short message service (SMS) message will be received by the phone, and then SPS enables the phone's data interfaces to connect to the corresponding server to retrieve notification data via the normal data network (either broadband cellular data connection or WiFi connection). Once the notification data has been retrieved, SPS will disable the data interfaces again to conserve energy during the phone's idle state.

Besides energy conservation, the proposed SPS has additional benefits. First, it enables a smartphone user to better control what idle state data traffic is allowed during the phone's idle state. This is especially useful for mobile users who have unlimited or cheap text messaging but limited data plan. Second, it also enables a mobile user to control how frequent she wants to receive notifications from the push server by specifying her preference in SPS. This will prevent the mobile user from being too much distracted by a burst of notifications.

Our contributions in this paper are:

• We have proposed an SMS push based scheme to conserve energy consumption when smartphones are in idle state, especially at bad reception areas. The scheme exploits the traditional cellular text messaging as a side channel in facilitating network data communication when the data interfaces are disabled.

• We have developed a prototype server and an Android client that controls the data interfaces and utilizes the SMS based push service proposed in this paper. Our prototype allows us to simulate the notification traffic between a server and multiple smartphone applications. This enables us to measure the energy saving obtained using the proposed scheme. In addition, we have developed a server and client that use the push service commonly employed by current Android applications, Google Cloud Messaging (GCM) [7]. In this way, we are able to measure how our proposed SPS compares against traditional push services in term of notification delay and energy consumption.

CHAPTER 2: BATTERY IMPACT OF DATA INTERFACES

In this section we examine several factors responsible for making the data interfaces one of the main sources of energy consumption in a smartphone.

Inherent Power Requirements

In order to sustain the data rates required by smartphone applications, wireless radios consume a significant amount of energy. To prevent radios from consuming an unacceptable amount of energy, Radio Resource Control (RRC) states were implemented [8]. These states allow the radio to remain turned on at all times and consume less power when there is no network traffic. The specifics of the RRC states are different among different cellular technologies and cellular carriers. However, the state machine shown in figure 2.1 illustrates the general RRC states: high power state, medium power state, and idle state.

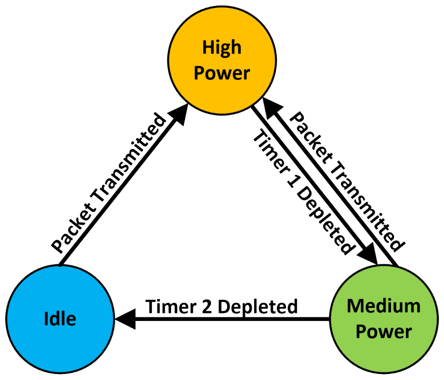


Figure 2.1: The RRC power states. The radio is in idle state when no data packets are transmitted, in high power state when data packets are transmitted, and in medium power state when transitioning from high power state to idle state.

Idle State. The default state of the radio in which it consumes the least power. While in this state, the radio periodically listens to the control channels for incoming voice calls and SMS messages. However, the radio is unable to send or receive data packets.

High Power State. The radio state when transmitting or receiving data packets. While in this state, the radio is provided the network resources necessary for fast data transmission such as dedicated uplink and downlink channels. As the name implies, this state consumes the greatest amount of energy of all the states.

Medium Power State. The medium power state is a transition state between the high power state and the idle state. It provides a middle ground between the low latency of the idle state and the large power consumption of the high power state. For example, in 3G UMTS networks it takes some time for the radio to transition from the idle state to the high power state. The medium power state allows the radio to remain at half power and, thus, take less time to reach the high power state. On the other hand, in 4G LTE networks the radio monitors periodically the control channel to initiate data transmissions. In the medium power state the radio monitors at a higher rate than in the low power state and, thus, reduces the latency of data transmissions.

The actual amount of energy consumed by the radio in each of the RRC states varies among different cellular technologies, cellular carriers, and radio manufacturers. In a particular 3G UMTS network, for example, the high power, medium power, and idle states consume 800 mW, 460 mW, and 0 mW respectively [2]. On the other hand, in a particular 4G LTE network the radio consumes 3500 mW, 1000 mW, and 15 mw during the high power, medium power, and idle states respectively [3].

Transitions from lower power states to higher power states are triggered by data transmissions. Transitions from higher power states to lower power states are triggered by inactivity timers. When the radio transmits or receives data packets, different timers begin counting down. If new packets are transmitted or received while the timers are counting down, the timers are reseted. On the other hand, if the timers reach zero, the radio transitions to the next lower energy state. The number of timers and their duration varies among different cellular technologies and carriers.

The use of timers to transition to lower power states introduces a state known as the tail power state [2]. After finishing transmitting, the radio remain in a higher power state in anticipation of further data packets. During this time, the radio continues to consume the amount of energy characteristic of the higher power state even though there is no data packets being transmitted or received. The tail power state has a significant impact on the total amount of power consumed by the radio.

Bad Reception

Bad reception provokes several side effects all of which can significantly increase the amount of energy the radio of data interfaces consumes. These side effects, discussed below, increase the amount of energy that a radio consumes even if the smartphone is in the idle state. Because the average smartphone experiences bad reception 47% of the day [5], the aggregate impact of these side effects significantly contribute to the overall power consumed by the radio.

Ping-pong Effect. Bad reception can prompt a smartphone to frequently dissociate and reassociate with base stations. This phenomenon, known as the ping-pong effect, prompts the smartphone to produces unnecessary association traffic. Even more damaging, bad reception can prompt a smartphone to frequently switch from one cellular technology, such as 3G UMTS, to another, such as 4G LTE. These switches among cellular technologies consume significantly more energy than switches among base stations [9].

Wireless Link Adaptations. The smartphone radio is designed to modify it modulation, coding, and/or power parameters in order to increase throughput and decrease bit error rates when experiencing bad reception [5]. In general, these parameters adjustments result in reduced data rates and higher transmission powers. As a result, the radio user more energy to transmitting the same amount of data.

Retransmissions. Bad reception can result in packet losses or packet corruption. This, in turn, triggers retransmissions at the link layer. If the reception is very bad, retransmission at the link layer might be unable to deliver the packets before a transport layer timeout occurs. Both of this scenarios prompt the radio to enter or stay in the high power state for longer than necessary and, thus, increases the amount of energy used by the radio.

Idle State Traffic

The average smartphone spends the majority of the day in the idle state waiting for user interaction. While in the idle state most of the smartphone components, such as the display and the processor, are placed in a low-power state or powered off. The radio, however, often continues to be used even while the smartphone is in idle state to accommodate the data traffic of some applications. In the average smartphone this idle state traffic accounts for 19% all the traffic produced by the smartphone [5].

Most of the idle state traffic provides no functionality to the user while the smartphone phone is in the idle state. Hence, it can be considered unnecessary network traffic caused by careless application design. However, some of the idle state traffic is used by the push mechanism that delivers notifications to smartphones. Without this necessary idle state traffic, notifications cannot be delivered.

CHAPTER 3: CURRENT SMARTPHONE PUSH TECHNOLOGIES

In this section we introduce the four methods commonly used to deliver unsolicited messages to smartphones: push services from manufacturers, private push services, polling, and short message service.

Push Services from Manufacturers

There are a number of services available to push data to smartphones. These services are offered by the companies behind the major smartphone operating system, Google, Apple, Microsoft, Blackberry, and Amazon, and are designed to work only with their respective devices [7] [10] [11] [12] [13]. Although, each of these proprietary push services have different features and capabilities, their basic architecture is identical. An application server that wants to push a notification to a smartphone sends the message to the push service. The push service, then, forwards the message to the smartphone though a long-lived TCP connection between the push service and the smartphone [14]. Figure 3.1 illustrates the basic architecture of push services.

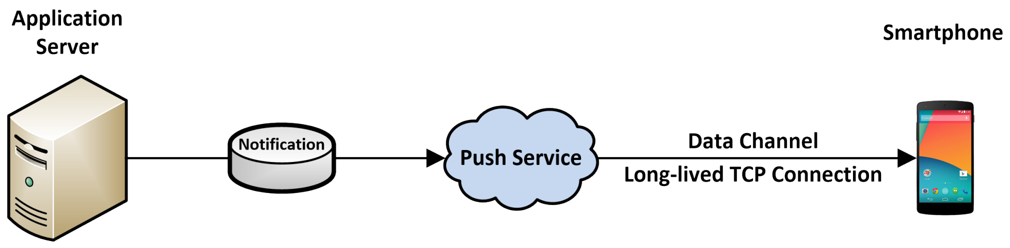


Figure 3.1: Push service architecture. To push a notification to a smartphone, the application server first sends the notification to the push service. The push service then delivers the notification to the push service client of the target smartphone using a long-lived TCP connection. Finally, the push service client delivers the notification to the target application.

The long-lived TCP connection between the push service and the smartphone is maintained by the push service client in the smartphone. The push service client is responsible for establishing a connection with the push service every time the smartphone connects to a data network. The push service client is also responsible for reestablishing the long-lived TCP connection every time the smartphone transitions from one network gateway to another, such as when the phone disconnects from 3G to connect to WiFi. All the applications in the smartphone that use the push service share the same long-lived TCP connection.

Before an application is able to receive push notifications from an application server, it must first register with the push service. To do this the application first requests an identifier from the push service. This unique identifier will later be used to route notifications to the intended smartphone and application. Once the application has received the unique identifier issued by the push service, it sends it to the application servers. The application server can then store the identifier and use it to push notifications to the application in the future. Figure 3.2 illustrates the push service registration process.

When an application server needs to push data to an application, it sends a notification with the unique identifier of the target smartphone to the push service. The push service then uses the unique identifier to identify the long-lived TCP connection with the target smartphone over which to send the notification. If there is no long-lived TCP connection currently open between the push service and the smartphone because the device is unreachable or turned off, the notification is stored in a queue for later delivery. The specifics of the push service queueing mechanism, such as the amount of time that a notification is stored before being dropped or the maximum number of notifications that can be queued, differs among the different push service providers. The technologies used to send notifications from the application server to the push service also depends on the push service provider. Google, for example, allows application servers to interact with their push service using HTTPS posts and XMPP [7].

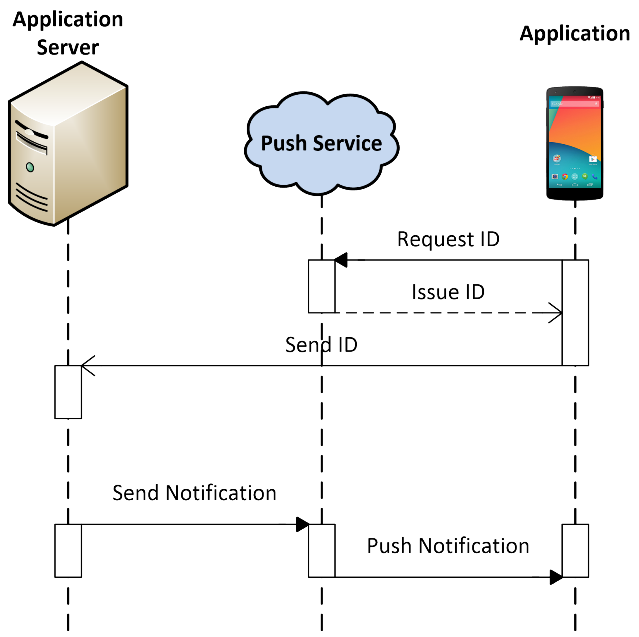


Figure 3.2: Push service registration process. The application first requests a unique ID from the push service which it then sends to the application server. The application server uses this unique ID to send notifications to the target smartphone using the push service.

After the push server client on the smartphone receives a notification from the push service, it uses the unique identifier to forward the notification to the intended application. The application can then take any appropriate actions. Push service providers support two types of notifications, send-to-sync notification and data notifications [7] [10] [13]. Send-to-sync notifications are small messages used to inform an application when there is new data available in the application server. The application can then connect with the application server directly to retrieve the new data. An email server, for example, could use a send-to-sync notification to inform an email application that a new email has arrived. The application could then retrieve the new email from the email server using a regular data connection. Data notifications, on the other hand, contain a payload. Applications that use this type of notifications can often acquire all the necessary data from the notification itself and, thus, do not need to sync with the application server. For example, a chat application could use data notifications to interchange messages among users. Because the user messages would be included in the payload itself, the chat application would not need to take any additional actions. Due to their small size, using send-to-sync notifications puts less strain on the battery than using data notifications.

Private Push Services

Instead of using the typical push service providers, some applications have implemented their own push service. This is most commonly seen among applications with a large number of users such as Facebook and Whatsapp [15] [16]. The technological principles of these private push services are identical to those of the typical push services. Notifications are sent to a middle agent which, in turn, uses a long-lived TCP connection to deliver the notification. The long-lived TCP connections are established and maintained using network protocols that support pushing notifications. The two protocols most commonly used to push notifications to smartphone applications are XMPP and MQTT.

XMPP (eXtensible Messaging and Presence Protocol) was originally developed in 1999 as an open standard for instant messaging applications [17]. Its flexibility, however, has lead to its expansion into a wide array of applications such as VoIP, videoconferencing, and cloud computing. XMPP uses open-ended XML streams to push data to smartphones usually over a long-lived TCP connection. However, Bidirectional-streams Over Synchronous HTTP (BOSH) can also be used to push data in scenarios where firewalls prevent communication using protocols other than HTTP. Some of the most prominent applications that use XMPP to push notifications are Skype and WhatsApp [18] [16].

MQTT (Message Queuing Telemetry Transport) was developed in 1999 for IBM's message-oriented middleware, WebSphere MQ [19]. MQTT was designed to allow WebSphere MQ to interact with remote devices over networks that are high latency, constrained, and high cost. Thus, is a lightweight protocol that introduces minimal overhead. MQTT pushes data to smartphones through a long-lived TCP connection maintained by a message broker. The most well known application that uses MQTT to push notifications is Facebook [15].

Implementing a private push service has several advantages over using a typical push service provider. First, it gives the developers of the application total control over the push service features. The private push service, thus, is able to use the notification queuing rules, notification size limitations, and service interface that best fits the application. Most importantly, however, the application server is not limited by the traffic volume restrictions imposed by typical push service providers. For example, the 40,000 messages per seconds restriction imposed by Google's GCM push service might not be enough for an application of the likes of Facebook. Finally, it allows the application to use the same push service in any platform. Application developers, therefore, do not need to change the notification modules of their application server to interact with applications running in different platforms.

Implementing a private push services introduces additional burdens on smartphone resources. Regular push service providers usually reuse a push mechanism necessary to the basic functioning of the operating system. For example, Apple's APNS push service reuses the long-lived TCP connection already used by the calendar and contacts applications of the smartphone [20]. Hence, using a private push service introduces an additional process that maintains another long-lived TCP connection between the smartphone and the private service. This could be avoided by using the regular push service provider. In addition, regular push service providers often employ a number of optimizations to improve battery life. For example, Google's GCM push service might briefly delay one notification to send multiple notifications at the same time and, thus, decrease the amount of time the smartphone's radio is in high power mode [21]. Since a private push service is isolated from other applications, this type of optimizations is not possible.

Polling

Although polling is not really a push mechanism, it is a method commonly employed by applications to retrieve unsolicited data from application servers. The POP3 email client protocol, for example, uses polling to retrieve new emails from the email server [22]. In some cases, polling can be an acceptable approach for retrieving unsolicited data, specially if the frequency of updates is known by the client in advance. For example, a weather application could effectively use polling to retrieve the weather forecast from an application server if it knows that the forecast will be updated every day at midnight. In addition, polling has two unique advantages over all the other push mechanisms discussed. First, it is the simplest approach for retrieving unsolicited data. Second, it does not require any additional resources to be implemented.

In many cases, however, the server update frequency is not known. For these cases, using polling on smartphone applications has several drawbacks. Polling has a large impact on the battery life of smartphones. A single application polling at a five minute interval can, on its own, drain over 10% of the battery [23]. Polling produces unnecessary network traffic that drains the data plan of smartphone users and, as a result, could induce additional costs. Finally, polling does not retrieve unsolicited data in real time. For example, if the polling interval of a given application is 15 minutes, and new data arrives at the server a second after the last poll was executed, the data will not be retrieved until the next poll is executed 14 minutes and 59 seconds later. Increasing the polling interval would reduce this problem. However, it would also intensify the battery consumption and unnecessary network traffic generated by the application.

Short Message Service

The Short Message Service (SMS) is among the oldest and most widely used methods of mobile communication. SMS was introduced along with phase 2 of the GSM standards in 1992 [24]. Due to its popularity, it was later expanded to be compatible with other popular cellular technologies such as CDMA and TDMA. Although SMS is most commonly associated with user-to-user text messaging, it can be employed for a variety of purposes such as weather alerts and news updates.

An SMS message interacts with a number of agents throughout its transmission process. All SMS messages to and from a cell phone go through a Short Message Service Center (SMSC). The SMSC is responsible for handling any retransmission if a message cannot be deliver. If the target cell phone is turned off or out of range, or if the network is not functioning, the SMSC can store the message and attempt a retransmission at a later time.

Before the SMSC can forward a message to the target cell phone, it must first find its location. The SMSC does this by querying the Home Location Register (HLC). The HLC is a database in which the information of all the cellular network subscribers is maintained. It contains phone numbers, service plan information, and other subscriber information. Most importantly, however, the HLC maintains the location of all the cellphones in the network. When a cell phone is first turned on, or when it moves from the coverage area of one base station to another, the cell phone updates its location in the HLC. Once the SMSC determines the location of the target cell phone by interrogates the HLC, it is able to route the SMS message to the correct Mobile Switching Center.

The Mobile Switching Center (MSC) is responsible for switching connections between the cellular network and the cell phones. Each MSC has a Visitor Location Register (VLR) which contains the precise location (cell) where the cell phone is located. The MSC first queries the VLR and transfers the SMS message to the correct Base Station System (BSS). The BSS then uses transceivers to send

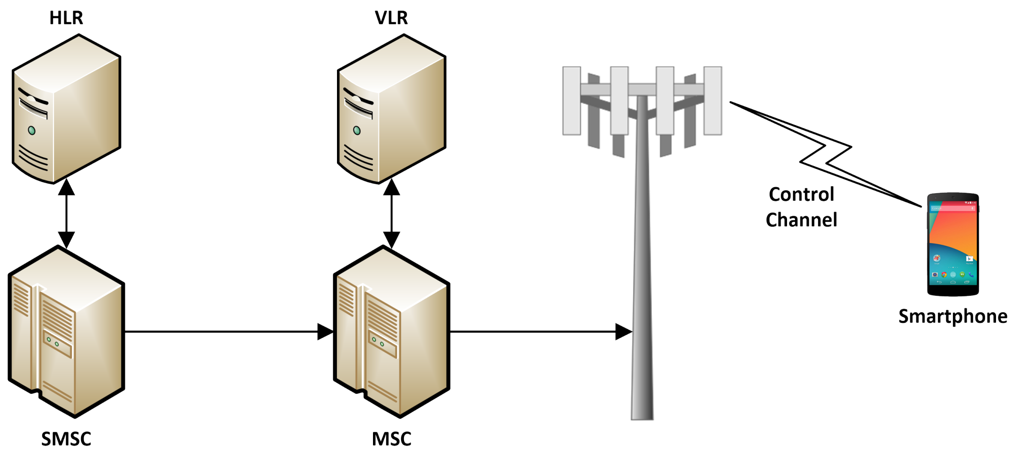


Figure 3.3: SMS architecture. The SMSC first queries the HLC to determine the general location of the target cell phone. Then, it sends the SMS message to the appropriate MSC which, in turn, queries the VLC to determine the exact location of the target cell phone. Finally, the MSC sends the SMS message to the appropriate base station. The base station then uses a control channel to deliver the SMS message to the target smartphone.

the SMS message to the target cell phone through a wireless control channel. Thus, a cell phone is able to send and receive SMS messages during a voice call or while using data services. Figure 3.3 illustrates the entire SMS message transmission process.

As specified in the GSM standards, SMS messages can be up to 140 bytes in size. Using 7-bit encoding, this amounts to 160 characters. The 140 bytes limit was set in part due to concerns over the available bandwidth at the time that the standards were drafted. Although the bandwidth available in modern cellular networks is more abundant, the low-bandwidth specifications of SMS have contributed to the universal adoption of the service.

SMS has several advantages over the other push technologies used in smartphones. The main of which being that it does not require the smartphone to be connected to a data network to receive push messages. The infrastructure necessary to use the service has been widely implemented. It is a proven technology capable of serving a large number of users. It already supports many of the features of typical push services. And, finally, it is a true push mechanism in which an unrequested message can be sent without the smartphone having to maintain a long-lived TCP connection.

CHAPTER 4: PROPOSED ENERGY SAVING SCHEME

In this section we describe our proposed SPS energy saving scheme. Basically speaking, it contains two parts: radio frugality policy, and SMS-based push service.

Radio Frugality Policy

We decrease battery impact of the wireless radio by reducing the amount of time the data interfaces are used throughout the day. We achieve this by disabling the data interfaces anytime the smartphone is not being used. That is, when the smartphone is in the idle state. And, enabling the data interfaces anytime the smartphone returns to the active state. Under this radio frugality policy, the user of a smartphone continues to receive phone calls and text messages when the phone is in idle state and is able to use the data service anytime he/she desires. Thus, from the user's perspective, data service appears uninterrupted. Because smartphones spend the majority of the day in the idle state [6], this policy significantly reduces the amount of time the radio is being used unnecessarily. The proposed radio frugality policy mimics the policies mobile platforms apply to other power hungry components such as the display and processor. These policies treat power hungry components as expensive resources and, thus, try to minimize the amount of time they are used.

The proposed policy saves energy by addressing two of the main sources of energy consumption in the smartphone radio when the phone is in idle state. First, it reduces the impact of bad reception in the radio. As discussed in Section II, bad reception prompts the radio to consume significantly more energy regardless of whether the smartphone is in the idle or active state. Given the fact that on average a smartphone experiences bad reception 47% of the day [5], this contributes significantly to the overall energy consumed by the radio. Since the average smartphone spends about

23 hours throughout a day in idle state [6], this radio frugality policy reduces the majority of bad reception side effects on the radio energy consumption.

Second, the proposed policy reduces the amount of unnecessary traffic produced by the smartphone. On average, 19% of all the traffic generated by a smartphone is produced during the idle state [5]. Most of this idle state traffic can be consider unnecessary because it provides no immediate functionality to the user. Idle state network traffic is often the result of careless application design, such as not taking the extra steps to consider the smartphone's state before generating network traffic. But also, it is the result of smartphone operating systems not enforcing stricter resource allocation policies for the data interfaces. In Android and iOS, for example, applications do not require a special set of permission to use the data interfaces while the smartphone is in the idle state. Thus, it is easy for a developer to make an application that mistakenly continues to produce network traffic even while the smartphone is in the idle state.

SMS-based Push Service

In current smartphone world, push services use long-lived TCP connections to create a "virtual circuits" from push servers to smartphones. Using these circuits, push services are able to send data that the smartphones have not requested. However, the data network interfaces of smartphones must remain on at all times to maintain the virtual circuits open. Thus, disabling the data interfaces while a smartphone is in the idle state would prevent notifications from being delivered to the smartphone. To allow smartphones to continue receiving notifications even while the data interfaces are disabled, we propose using SMS as a side channel to facilitate the delivery of notification data. Because SMS uses traditional cellular technology to route and deliver messages [24], it enables the push service to inform a smartphone of incoming notification even when the phone's data interfaces are disabled.

The first step for an application to receive notifications through the proposed SMS push service is to register the phone's number with the application server. This can be easily done when the app is installed and connects to the application server for the first time. The application server then stores the phone number for future use. Once the application server has new data for the application, it simply sends a wakeup SMS message to the smartphone using the phone number received during the registration process. The SMS message may contain an optional but small payload (e.g., the ID of the application, type of the notification message, or the complete notification message itself if it can be fit into one SMS message), or contains no payload at all. Upon receiving the SMS wakeup message, the SPS client code on the smartphone will enable the data interfaces (3G/4G cellular and WiFi) to synchronize with the server using a data connection (or, if the SMS message payload contains all the necessary information, assimilate the message directly). Figure 2 shows the complete procedure of SPS.

Besides allowing the data interfaces to be disabled while the smartphone is in idle state, the SMS push is inherently more energy efficient than typical push services. First, maintaining the virtual circuit between the smartphone and the server required by traditional push services introduces additional idle state traffic. It requires the smartphone to continue to establish a long-lived TCP connection with the push service every time the smartphone loses and regains connection, switches wireless data technology, and changes gateway. In addition, it requires periodic transmissions of keep-alive packets to prevent firewalls or NATS from dropping the long-lived TCP connection [25]. Because SMS uses traditional cellular technology to route and deliver messages, the proposed SMS push does not produce any idle state traffic. Also, SMS messages are delivered through control channels. As a result, the radio does not enter the high power state when receiving an SMS notification [8].

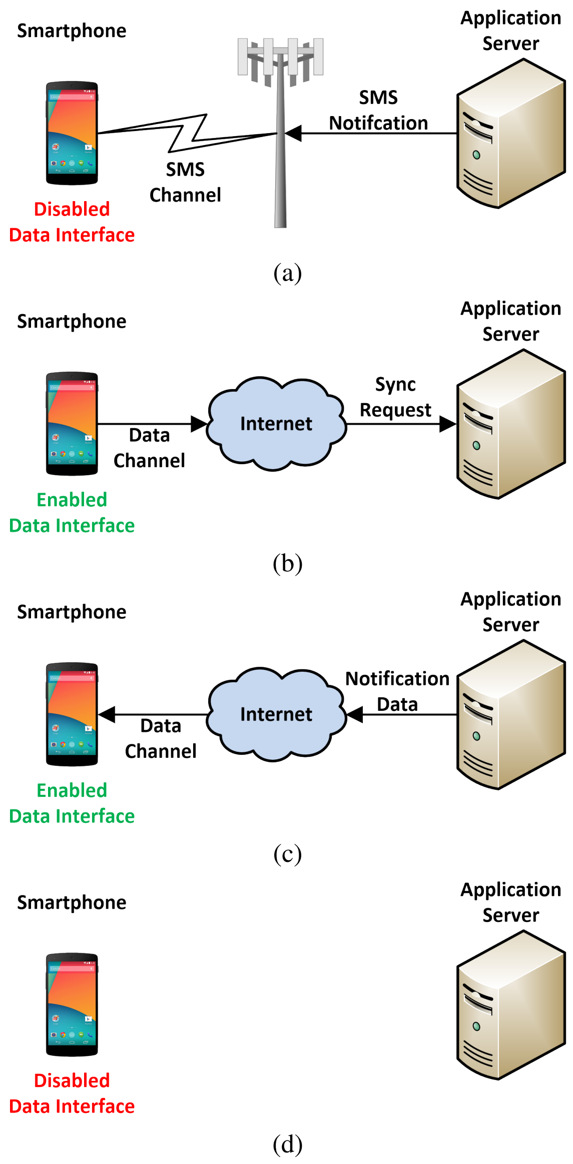


Figure 4.1: SMS-based push service procedure for the proposed SPS in phone's idle state. (a) When the application server has new data for the smartphone and the phone's data interfaces are disabled, it first sends an SMS message to the smartphone. (b) The SMS notification prompts the smartphone to enable its data interfaces and send a sync request message to the application server using the regular data service. (c) The application server replies to the sync request by sending the new data to the smartphone. (d) Once the smartphone receives the new data, it disables its data interfaces again. When the phone is in active state, the SPS still works by skipping the procedure in (d), i.e., does not disable the data interfaces.

CHAPTER 5: EVALUATION

To evaluate the proposed SPS energy saving scheme, we have developed a server and an Android client prototype. The source code of the server and client used throughout the evaluation is available online at `http://www.cs.ucf.edu/~czou/SPS`.

Prototype

The SPS server was implemented in Java and is able to send sync notifications to the target smartphone using SMS. There are multiple methods to send SMS messages programmatically. In our prototype, we use the SMTP SMS gateways provided by the carriers. Our prototype server first connects to a Gmail server and uses it as a proxy to send an SMS message to the SMTP SMS gateways. This is done by simply sending an email to `phonenumber@carrierdomain.com`. We use this method in our prototype because it is cost free.

The SPS client is implemented as an Android application that automatically begins running in the background when the smartphone is powered on. The SPS client is responsible for both enforcing the proposed radio frugality policy and handling the SMS-based push service. The SPS client automatically disables or enables the data interfaces (both 3G/4G and WiFi) every time the smartphone enters the idle or active state respectively. In addition, the SPS client monitors all incoming SMS messages for SPS notifications. If an SPS notification text message is received, the client begins the server sync process. To do this, the SPS client enables the data interface, retrieves the data from the application server using a TCP connection, and disables the data interface. In addition, the client discards the SPS text message. This prevents the SPS notification text message from reaching the Android text messaging application and, thus, notifying the user every time a SPS notification is received.

Throughout our experiments the SPS server runs in an Amazon AWS server and the SPS client runs in an LG MS840 Android smartphone with a 1,540 mAh battery. The LGMS840 smartphone is able to use 1xRTT CDMA, EVDO, and LTE network technologies throughout the experiments. In addition, we have implemented a test application and a test application server that uses the current Google's GCM push service [7]. We use the GCM implementation to compare how the proposed SPS performs against traditional data channel based push services.

Finally, it is worth mentioning that although we only use sync notifications in all the experiments, our prototype does support data notifications purely using SMS channel without data channel involved. In this mode, as long as the notification data is not big, it can be directly encoded into one or several SMS messages sent to the smartphone, where the SPS client absorbs notification data without any further action. Hence, we believe this implementation is trivial and does not provide much insight in measuring the battery impact of the proposed radio frugality policy and SMS push service. However, such notification method would consume even less energy than SPS sync notifications because the data interfaces do not need to be enabled after receiving an SMS message.

Battery Usage

To measure the battery impact of the proposed scheme, we simulated notification traffic and use the SPS and the GCM implementations to handle it. In addition, we define two types of users to study how the number of notifications affects push service energy consumption. The first type, the social user, is defined as a Facebook, WhatsApp, and email user. The second type, the business user, is defined as an email user. Given that on average Facebook, Whatsapp, and email applications receive 82, 116, and 105 notifications throughout the day [26] [27] [28], we define the social user and business user notification traffic as on average 303 and 105 notifications per day respectively. We simulate the notification traffic using a Poisson process, and assume that all notifications are received within a 16-hour time window in each day.

We run the social and business user notification traffic simulations for 3 hours using both the SPS and GCM implementations. In addition, we perform each experiment in an area with bad reception, where the signal strengths for 1xRTT/EVDO and LTE are on average -110 dBm and -130 dBm respectively, and an area with good reception, where the signal strength for 1xRTT/EVDO and LTE are on average -85 dBm and -94 dBm respectively. Throughout each simulation we record the battery's state of charge. The state of charge is calculated by the Android operating system to indicate how much of the battery's capacity has been used. Figure 5.1 shows the result.

Figure 5.1 shows that SPS consumes significantly less energy than GCM when the smartphone is in bad reception area. In fact, given our test environment, the battery of a smartphone using SPS would last twice as long as that of a smartphone using GCM.

Figure 5.1 also shows that SPS consumes less energy than GCM when the smartphone is in a good reception area. Also, it shows that notification traffic volume slightly affects the amount of energy consumed by the smartphone when using both SPS and GCM. However, we can conclude that most of the energy consumption in idle state is caused by bad reception, not by the volume of data traffic, and the proposed SPS works best if a smartphone stays a long period in bad reception areas throughout a day.

Delay

We measure the delay of delivering a notification using SPS against GCM. The results are shown in Figure 5.2. In average, it takes on average 9.8 sec to deliver a notification using SPS. On the

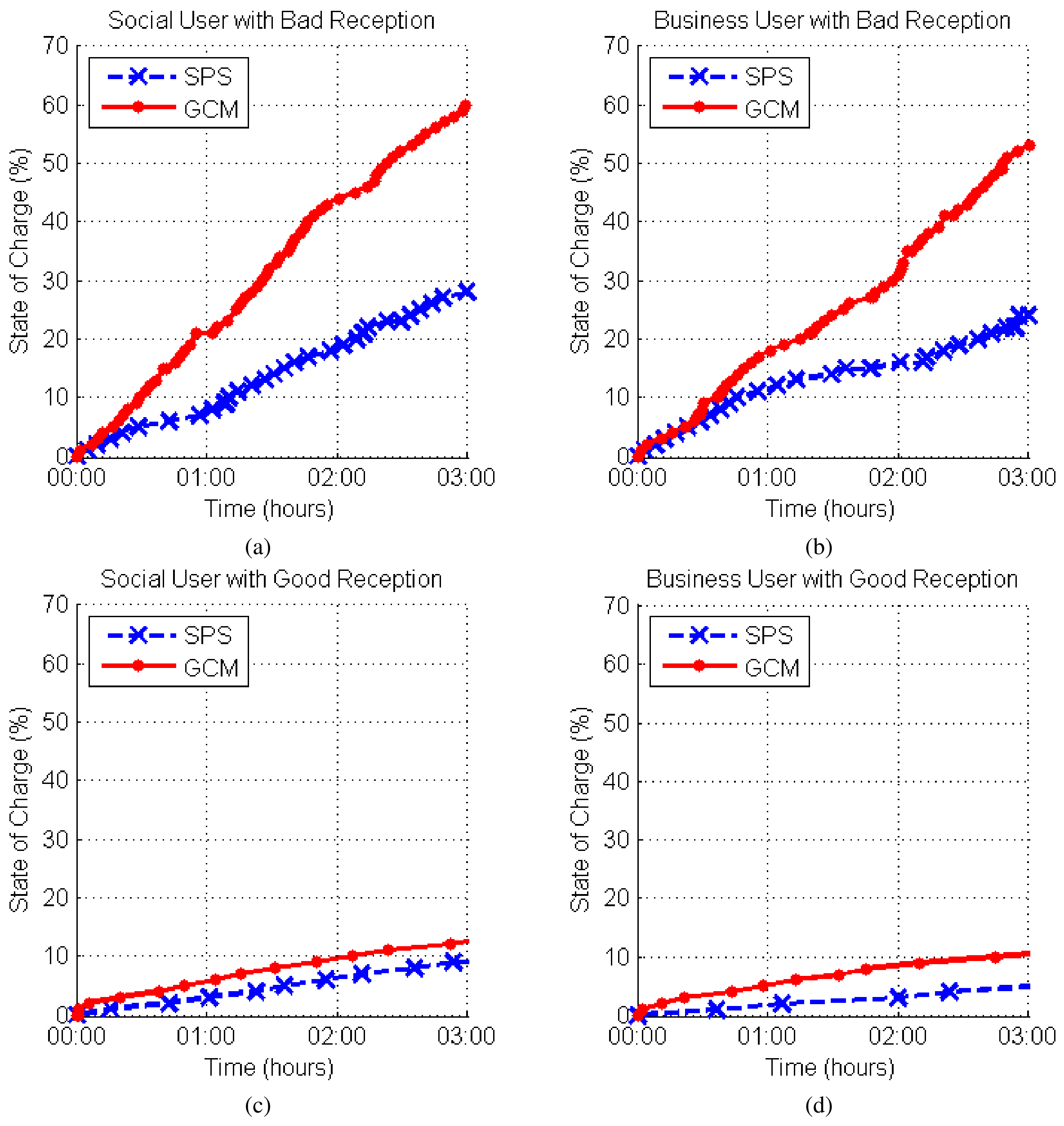


Figure 5.1: Battery charge used by SPS vs GCM of a social (a) and business (b) user experiencing bad reception. Battery charge used by SPS vs GCM of a social (c) and business (d) user experiencing good reception.

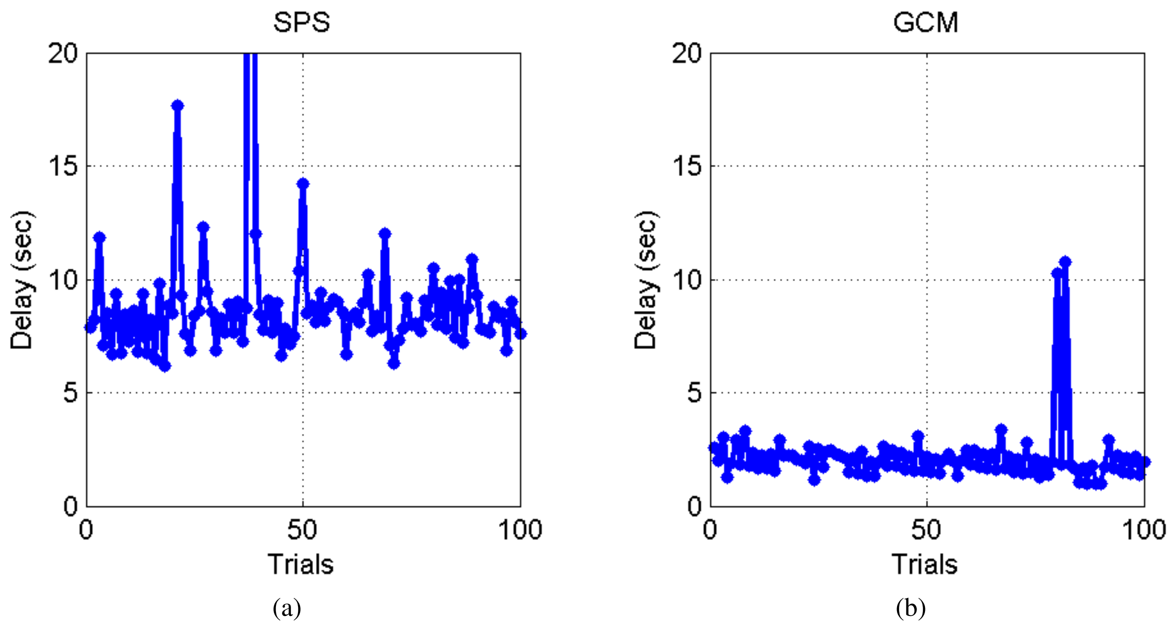


Figure 5.2: The delay to push a notification using SPS (a) and GCM (b).

other hand, it takes on average 2.1 sec to deliver notification using GCM.

We further analyze the time delay of SPS notifications by recording each event that occurs on the prototype server and application. To ensure that our measurments are accurate, we synchronize the smartphone and the servcer using a `pool.ntp.org` time server [29]. Using this information, we can split the SMS push notification delay into three parts:

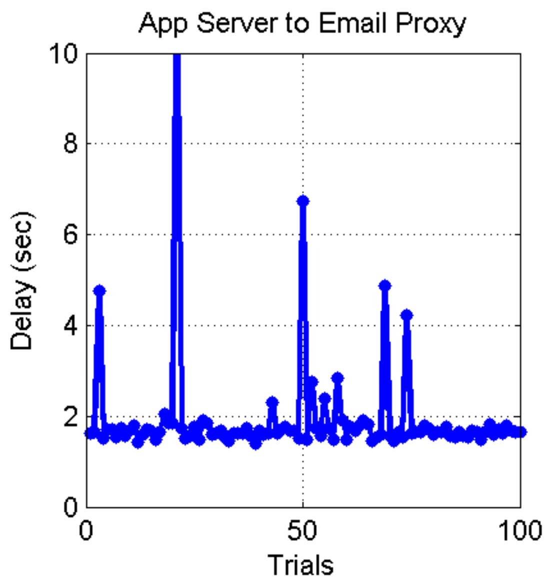
• The application server sends an email to the email proxy;

• The email proxy sends SMS message to the smartphone;

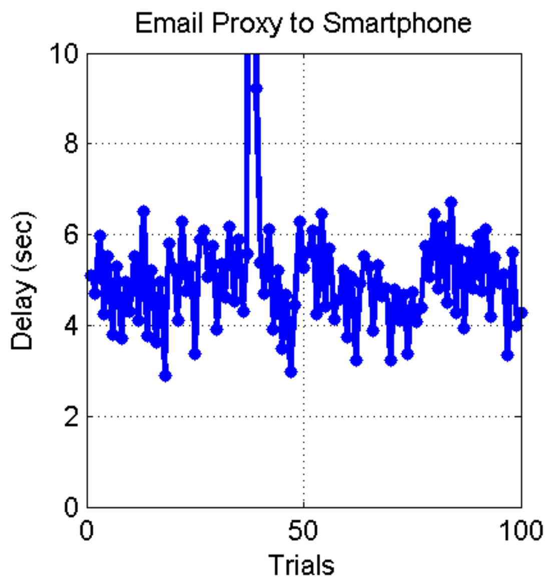
• The smartphone activates data interfaces to retrieve notification data from the application server.

Figure 5.3 shows the time delay for the three parts in SPS notification. On average, 1.9 sec are spent in sending the notification to the proxy email server used in our cost-free implementation; and 6.2 sec is spent by the cost-free email proxy to complete SMS message delivery. There are alternative methods to send SMS notifications without involving an SMTP gateway. These methods could reduce the SPS delay significantly by removing the email proxy and the carrier's SMTP server from the notification transmission procedure. We used the SMTP SMS gateway method in our prototype because it is provided by the cellular carriers free of cost. The alternative faster methods require additional hardware and/or paying service fees.

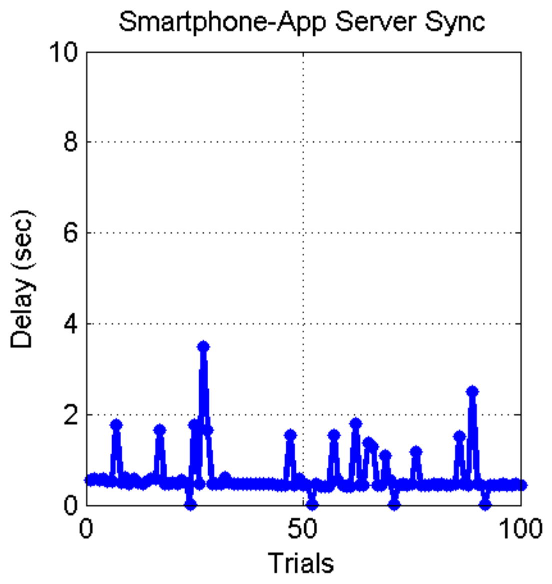
In addition, we observe that the extra step of enabling the data interfaces before retrieving the notification data adds minimal delay. On average, it takes 630 msec for SPS to sync with the application server while GCM takes 487 msec.



(a)



(b)



(c)

Figure 5.3: SPS delay (a) Delay between the application server and the email proxy. (b) Delay between the email proxy and the smartphone. (c) Delay to retrieve the new data from the application server.

CHAPTER 6: RELATED WORK

Previous works on the impact of the cellular data interface in the battery life of smartphones have focused on the RCC power states [2] [3] [4]. These works conclude that the RCC state machine introduces significant energy inefficiencies because of the power state promotion overhead and the tail effect. In order to reduces the amount of energy consumed by the smartphone's radio, these works propose modifying the inactivity timers used to transition from high power states to low power states. Specifically, these works propose optimizing the static inactivity timer values currently used or implementing dynamic inactivity timers that are shaped by the network traffic. Our work is different from these works because it addresses external factors, such as unnecessary idle state traffic and bad reception, that significantly contribute to the overall energy consumed by the radio.

Zhang et al. proposes E-MiLi (Energy-Minimizing idle Listening) the reduce energy consumption of Wi-Fi components [30]. By analyzing real-world Wi-Fi traffic traces, the author determines that idle listening is responsible for 60% to 80% of the Wi-Fi's energy consumption. To reduce the energy consumption during idle listening, E-MiLi reduces the clock-rate of the radio during idle listening and reverts to full clock-rate when the radio transmits or receives packets. In addition, E-MiLi introduces a novel detection scheme that allows the receiver to detect incoming packets even if its sampling rate is much lower than the bandwidth of the transmitting signal. Our work is different from Zhang's because our proposed SPS reduces the Wi-Fi's energy consumption in smartphones caused by external factors such as bad reception and unnecessary idle state traffic.

Ding et al. investigate how bad reception increases the amount of energy consumed by the smartphone radio [5]. The author quantifies the amount of energy used by a radio experiencing bad reception by conducting experiments under control conditions. Using this information, along with reception traces collected from 3785 volunteers, the author is able to develop a more accurate model of smartphone battery life. Furthermore, Ding proofs that bad reception significantly increases the amount of energy that the smartphone radio consumes. In short, bad reception prompts the smartphone radio to stay in the high power state and in the tail power state for a longer time. Using the new smartphone energy consumption model, Ding concludes that buffering idle state traffic while the smartphone is experiencing bad reception could reduce energy consumption by up to 21.5%. Our approach is different from Ding's because it allows the smartphone to continue to receive notification even if there is bad reception. In addition, our approach also addresses energy consumption due to unnecessary idle state traffic.

There are several works that study energy consumption of the push mechanisms employed in smartphones. Haverinene et al. and Gupta et al. investigates how the keep-alive traffic used by push mechanisms affects the battery lifetime of smartphones in WCDMA and LTE networks respectively [25] [31]. Both authors propose modifying the RCC parameters to reduce the impact on keep-alive traffic in the battery life. Meng et al. proposes using buffers and a number of middle agents to reduce energy consumption of push mechanism used by instant messaging applications in smartphones [32]. Dinh et al. compares the energy consumed by an application that uses push services against an application that uses polling to retrieve unsolicited data from a server [23]. Our work is different from all these works because, rather than fine tuning the current push mechanism employed in smartphones, it proposes using a completely different push mechanism based on SMS channel instead of traditional data channel.

CHAPTER 7: CONCLUSION

In this paper we propose a new energy saving scheme to reduce unnecessary power consumption when smartphones are in the idle state, especially when they are in bad reception areas. First, our scheme implements a policy that disables the data interfaces every time the smartphone enters the idle state and re-enables it every time the smartphone enters the active state. By doing this, the proposed policy is able to neutralize two of the main sources of energy consumption related to the data interfaces: bad reception and unnecessary idle state traffic. To support real-time notification functionality required by some applications, we propose a notification system that relies on SMS to temporarily enables the data interfaces to retrieve notification data through normal data channel. The proposed notification system allows smartphones to continue to receive notifications even if the data interfaces are disabled.

We developed a prototype of the proposed energy saving scheme and we tested in a realistic environment. Our evaluation shows that our approach consumes significantly less energy than the current approach. In fact, using our system the battery of the smartphone lasts twice longer than using the current system when the smartphone is experiencing bad reception.

LIST OF REFERENCES

[1] J. Heggestuen. (2013) One in every 5 people in the world own a smartphone, one in every 17 own a tablet. [Online]. Available: http://www.businessinsider.com/smartphone-and-tablet-penetration-2013-10

[2] F. Qian, Z. Wang, A. Gerber, Z. M. Mao, S. Sen, and O. Spatscheck, "Characterizing radio resource allocation for 3g networks," in *Proceedings of the 10th ACM SIGCOMM conference on Internet measurement*. ACM, 2010, pp. 137–150.

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

[4] H. Falaki, D. Lymberopoulos, R. Mahajan, S. Kandula, and D. Estrin, "A first look at traffic on smartphones," in *Proceedings of the 10th ACM SIGCOMM conference on Internet measurement*. ACM, 2010, pp. 281–287.

[5] N. Ding, D. Wagner, X. Chen, Y. C. Hu, and A. Rice, "Characterizing and modeling the impact of wireless signal strength on smartphone battery drain," in *Proceedings of the ACM SIGMETRICS/international conference on Measurement and modeling of computer systems*. ACM, 2013, pp. 29–40.

[6] J. Fetto. (2013) Americans spend 58 minutes a day on their smartphones. [Online]. Available: http://www.experian.com/blogs/marketing-forward/2013/05/28/americans-spend-58-minutes-a-day-on-their-smartphones/?WT.srch=PR_EMS_smartphones_052813_press

[7] Google cloud messaging for android. [Online]. Available: http://developer.android.com/google/gcm/index.html

[8] M. Sauter, *3G, 4G and Beyond: Bringing Networks, Devices and the Web Together*. John Wiley & Sons, 2012.

[9] A. Pathak, Y. C. Hu, and M. Zhang, "Bootstrapping energy debugging on smartphones: a first look at energy bugs in mobile devices," in *Proceedings of the 10th ACM Workshop on Hot Topics in Networks*. ACM, 2011, p. 5.

[10] (2014) Apple push notification service. [Online]. Available: https://developer.apple.com/library/ios/documentation/NetworkingInternet/Conceptual/RemoteNotificationsPG/Chapters/ApplePushService.html

[11] (2014) Push notifications for windows phone. [Online]. Available: http://msdn.microsoft.com/en-us/library/windowsphone/develop/ff402558%28v=vs.105%29.aspx

[12] (2014) Push service. [Online]. Available: https://developer.blackberry.com/services/push/

[13] Amazon device messaging. [Online]. Available: https://developer.amazon.com/appsandservices/apis/engage/device-messaging

[14] C. Miller, D. Blazakis, D. DaiZovi, S. Esser, V. Iozzo, and R.-P. Weinmann, *IOS Hacker's Handbook*. John Wiley & Sons, 2012.

[15] (2013) Examples of mqtt usage. [Online]. Available: http://mqtt.org/wiki/doku.php/example_uses

[16] M. Garca. (2013) Drupal whatsapp. [Online]. Available: https://drupal.org/project/whatsapp

[17] P. Saint-Andre. (2004) Extensible messaging and presence protocol (xmpp): Core. [Online]. Available: http://www.ietf.org/rfc/rfc3920.txt

[18] J. Roettgers. (2011) Skype adds xmpp support, im interoperability next? [Online]. Available: http://gigaom.com/2011/06/28/skype-xmpp-support/

[19] Frequently asked questions. [Online]. Available: http://mqtt.org/faq

[20] (2013) Well known tcp and udp ports used by apple software products. [Online]. Available: http://support.apple.com/kb/ts1629

[21] Gcm advanced topics. [Online]. Available: http://developer.android.com/google/gcm/adv.html

[22] J. Myers and M. Rose. (1996) Post office protocol - version 3. [Online]. Available: http://www.ietf.org/rfc/rfc1939.txt

[23] P. C. Dinh and S. Boonkrong, "The comparison of impacts to android phone battery between polling data and pushing data," in *IISRO Multi-Conferences Proceeding. Thailand*, 2013.

[24] K. Amri and T. Ceglarek. Sms: How does it work? [Online]. Available: http://services.eng.uts.edu.au/userpages/kumbes/public_html/ra/sms/

[25] H. Haverinen, J. Siren, and P. Eronen, "Energy consumption of always-on applications in wcdma networks," in *Vehicular Technology Conference, 2007. VTC2007-Spring. IEEE 65th*. IEEE, 2007, pp. 964–968.

[26] The average facebook user receives 82 notifications everyday. [Online]. Available: https://twitter.com/UberFacts/status/263487232613163008

[27] S. McGlaun. Whatsapp handles 50 billion messages daily, more than sms delivery. [Online]. Available: http://www.slashgear.com/whatsapp-handles-50-billion-messages-daily-more-than-sms-delivery-21313892/

[28] Q. Hoang. Email statistics report, 2011-2015. [Online]. Available: http://www.radicati.com/wp/wp-content/uploads/2011/05/Email-Statistics-Report-2011-2015-Executive-Summary.pdf

[29] How do i use pool.ntp.org? [Online]. Available: http://www.pool.ntp.org/en/use.html

[30] X. Zhang and K. G. Shin, "E-mili: energy-minimizing idle listening in wireless networks," *Mobile Computing, IEEE Transactions on*, vol. 11, no. 9, pp. 1441–1454, 2012.

[31] M. Gupta, S. C. Jha, A. T. Koc, and R. Vannithamby, "Energy impact of emerging mobile internet applications on lte networks: issues and solutions," *Communications Magazine, IEEE*, vol. 51, no. 2, pp. 90–97, 2013.

[32] L.-S. Meng, D.-s. Shiu, P.-C. Yeh, K.-C. Chen, and H.-Y. Lo, "Low power consumption solutions for mobile instant messaging," *Mobile Computing, IEEE Transactions on*, vol. 11, no. 6, pp. 896–904, 2012.