EnergyMAC : A Framework for User-centric Energy Control

Harsha and Satish

I. Problem

One of the main requirements for an Android application is energy efficiency. Current energy management techniques in android give a little scope for energy control of applications based on user preferences. In this project, we develop a framework based on Android to enable user-centric energy management. This framework will enable users to give priority to the energy consumption of apps that are of high value to them. Furthermore, it will also give incentive for developers to create applications that are energy efficient.

There are three stakeholders that are mainly affected by this framework

1) Android end users
2) Android App developers
3) Smartphone device manufactures

Users have a limited power in their Smartphone batteries. Among many applications installed by a user on their Smartphone, only a few frequently used applications are more important for them. For a user who is mindful of his battery life may want to restrict applications that have a high energy usage but has a little value for the user. To restrict such apps from consuming all of the energy, a user can use our framework to provide a priority for each app installed on their Smartphone. A user can give either high, medium, or low priorities for an app based on their preference. By default, applications will be in low priority when they are initially installed in the phone.

High priority applications and the Android operating system itself are considered to be most important applications for the user and are allowed to unrestricted use of the energy. Energy usage of medium and low priority applications are considered to be less important for the user and energy usage of these applications are restricted by our framework.

Android application developers should adapt their lower priority applications to make them more energy efficient. Energy used by applications will be accounted for and certain system resources will be denied when they use more energy than allowed by its priority. Application developers should accommodate these new insufficient energy errors and build their app accordingly to work for low, medium and high priorities. Some of the current applications such as Facebook have both full-fledged and lite applications for different kinds of phones and networks.

Accurate energy accounting depends on specific device and battery used. Hence the framework should be customizable for device manufacturers so that they can optimize for their devices and battery type used.

Keeping various stakeholders in mind, our system should always ensure following properties

1) Always higher priority applications can use more energy than a lower priority application.
2) Fairness among all applications belonging to same priority should be ensured.
3) The system should consider the changes in total available energy as the battery drains over the time.
4) It should also take into account for changes in the number of energy consumers that are active in the Smartphone.
5) The framework should have minimal overhead.
6) The framework should ensure failures due to lack of energy are graceful and developers are notified.
7) The framework shouldn't reclaim energy once allocated to an app unless application releases or gets killed.
8) Apps should benefit when system gains energy due to recharge of battery.
9) The system should be customizable for different hardware devices and battery.

In the next section, we will look at how system is designed to satisfy these properties.

II. Solution

A. System design

In this section, we will introduce fundamental unit of measurement for energy in our system, how we account for energy consumption of applications and finally how we will allocate energy between applications to ensure system properties are met.

B. Unit of measurement

An abstract unit of energy in our system is called energy credit. Having an abstracted unit will help us to make the framework more customizable for Android device makers. By default, an energy credit is equivalent to 1 mAh in the current prototype implementation. When a resource of an Android system such as the network or the disk is used by an application, it will spend some energy credits equivalent to the amount of energy spent in executing that system functionality.

C. Energy accounting for system resources

Android uses Linux kernel to manage system resources. In Linux kernel, system calls are the single point of contact for all user-level applications. Any user functionality that requires system resources such as network can only be executed by calling corresponding system calls. Using this fact, there are many previous android energy modeling techniques such as Eprof, which can calculate the energy consumption for each system call. In our framework, we will estimate amount of energy required to execute a system call and allow system call to execute only when required energy for application is available.

System calls can have either constant or varying energy cost. System calls such as socket::connect has a constant energy cost and socket::send has an energy cost depending on the amount of data it needs to send. Constant cost system calls can be estimated before hand and can be stored in a hash table. Verifying system calls with dynamic cost require a model to predict cost based some variables such as amount of data to be transfered as in the case of socket::send. These models can be different for different devices, we can create these models using values estimated using Eprof based on different parameters.

D. Allocation of credits

Energy credit allocation among applications in a dynamic system such as Android can be tricky. The total number of available energy credits change as the battery is discharged continuously. Number applications consuming energy will also change with time. Furthermore, some applications will only be installed in the system, but may never be used. Hence it's important to carefully allocate energy credits to ensure all of the system properties stated in the section **[?]** are met.

The lithium-ion batteries used in modern Smartphones have the capacity range from 1000 mAh to 5000mAh. For example, If a phone is fully charged and the battery has a capacity of 2000mAh and if applications has a continuous load of 200 mA, then the battery will last for 10 hours. By decreasing load on the battery, we can decrease overall power consumption and increase battery life. In the previous example, if applications only has a load of 100 mA then the battery will last for 20 hours. To facilitate this we need to limit the amount of energy spent by applications. In our framework, we will limit number energy credits an low priority application can spend in a time period. In the prototype implementation, we had the time period as 1 min.

Our system should limit energy consumption of low and medium android applications without affecting the energy consumption of the system and high priority applications. To ensure this, we will allocate an infinite number of credits for high priority and system apps. For medium and low priority applications, we will allocate 0.1% and 0.05% of total available credits respectively per time period.

E. Example

Below is the example on how it will work Battery capacity is measured in Milli-Ampiers per hour. Normal smart phone battery will have 2000 mAh as capacity for fully charged battery. If operations as specified in Pathak et al.

F. How we met system properties

1) This system will ensure that lower priority applications always has limited energy usage. This will force applications developers to create applications to have a minimal energy consumption.
2) It will always allocate applications with same priority same amount of energy credits.
3) Rate is proportional to amount of battery available

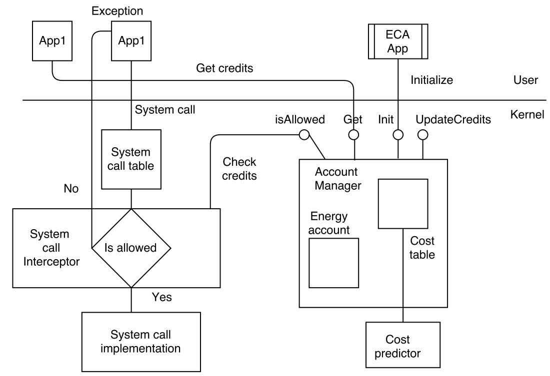

Fig. 1. Architecture.

G. Limitations

1) The number of credits required for a system call is approximation of original value.
2) Battery characteristics such as charge will change over the lifetime of the battery.
3) Certain existing operations which require high energy may fail.
4) In cases such as tail energy applications can have energy impact without calling system calls.

H. Assumptions

Following are assumptions we make in the Android system

1) We account for all energy consumption with system calls.
2) Apps either run in background or foreground are considered active and consume energy.
3) Inactive apps don't consume energy.
4) The total amount of available energy is constant for the time period.

III. Implementation

A. Technical Approach

Our EnergyMAC framework (see Fig 1) comprises of four components:

1) Energy Credits allocation System App (ECA app): is a system app that facilitates user to see and allocate priorities for user apps installed in the mobile device. When a user allocates the priorities for apps, ECA app invokes the system call of Energy Accounting Manager in android kernel to save the app details and designated credits.
2) System Call Interceptor: is a component that intercepts the system calls from apps and consults Energy Accounting Manager to check credits that will be spent for that System Call and credits remaining for the app. If the

enough energy credit is available, then it redirects to the actual system call else custom exception is thrown stating that required energy credits are not available for that operation.

3) Energy Accounting Manager: Two hash tables for storing records related to app's priority, assigned credits, and each System call's estimated cost (constant cost) are created in the Kernel. One table i.e. Energy account, is used for storing the app UID, priority and assigned credits and another table named Cost sheet is used for storing System call info and corresponding constant cost in terms of credits. Energy account will be refilled with credits at the end of each time period using a Linux kernel timer.

   Energy Accounting Manager has access to above tables and exposes APIs for various operations:
   - Get priority and length of time period for app/s
   - Update priority for app/s
   - Get cost of system call
   - Update cost of system call
   - Check if a given system call is allowed
4) System Call Price Manager: initializes cost table with cost for all system calls that have constant cost. Moreover, it will also estimate cost of variable cost system calls using models. These models will be derived from previous work such as Eprof.

B. How it works

Android OS is divided between User Space and Kernel Space based on the type of actions that can be performed by user program and OS itself. This separation is required to provide isolation and abstraction. The Kernel has the privileges for managing the system resources and User programs do not. So, user program communicate with Kernel via system call to use services provided by OS.

Each system call is represented by a number in system call table in the kernel. Since apps in the user space invoke system call for different operations, this system call can be intercepted and decision can be taken whether the system call should be executed. This is the basis for enforcing the energy accounting policies on apps in our energy accounting system.

In Energy accounting system, the user allocates the available priorities for the apps that user intends to use via ECA App. This internally invokes the custom system calls that are part of Energy Accounting Manager, to update the priority of the apps and data is stored in the Energy account hash table. The accounting system had a kernel timer which times out at the end of each time period and credits will be updated in Energy account hash-table based on the priority of the app for next time period.

The cost of each system call in terms of energy credits is decided by the System Call Price Manager depending upon the energy model for that system call and constant costs will be stored in the hash table named Cost sheet.

When an app, in this case, GPS App, tries to use get the location, then it invokes the system call to get the location of the device. At this time, the system call is intercepted by the Interceptor and it checks with the Energy Accounting Manager if an app has enough credits to actually invoke that system call related to GPS. If the app has credits more than required to invoke the system call, then the Interceptor redirects to the actual system call and the corresponding credits for that system call is deducted from the allocated credits for that app. Otherwise, the custom exception is thrown back to the application stating that not enough credit is available for that operation and app has to adapt itself accordingly.

The app can also check whether the operation is permitted and also time period with Energy Accounting Manager and change App's behavior accordingly.

C. Implementation choices

There are mainly 5 different layers in Android OS i.e. System Apps, Java API framework, Native Libraries and Android Run time, Hardware Abstraction Layer and Linux Kernel. The Energy Accounting system could have been part of any layer but the decision to make it part of Kernel for the following reasons:

1) The operations will be protected: The malicious apps will not be able to steal the energy credits or change priority of other benign apps. Only System app and kernel components will have access to the system call related to energy credits update APIs.
2) If it were pushed to any layer above kernel, that would have involved changes in multiple components in that layer and it would not have been easily extendibles for future changes. The kernel is the centralized place to receive all the system calls and due this the changes required in one place and less invasive. For example if this energy accounting were to be moved to the Java API framework level, closer to the layer where app reside, then the changes had to be done in multiple components like Location Manager, Telephony manager,etc in this layer and that would have been more invasive.
3) Previous work such as Eprof has models for calculating energy costs of system calls.

Following changes were made in OS Kernel as part of the implementation:

1) System calls and their implementation corresponding to the APIs of the Energy Accounting Manager were added to the Kernel.
2) Two hash tables were created in the kernel memory to store energy accounting related information.
3) The system call number of the GPS system call i.e. sys ioctl was replaced by the system call number of the Interceptor in the system call table in the kernel.

IV. Future work