QoE-centric Mobile Operating System Design
Scott Haseley and Geoffrey Challen
University at Buffalo - Department of Computer Science and Engineering

Smartphone Quality of Experience

Users care about how mobile operating systems manage human-facing resources, such as time, battery life, and network data. The management of these resources contributes to a smartphone user’s quality of experience.

While current operating systems are adept at managing hardware resources such as CPUs, disks, and memory, there is a lot that must be redesigned to quantify and improve QoE.

QoE-centric Design Principles

To meet smartphone users’ expectations, it is necessary to design systems that can accurately measure and understand QoE, and make decisions based on QoE. QoE-centric operating systems should:

- Accurately measure QoE
- Understand the contributions of various resources to QoE
- Continuously prioritize resources based on QoE
- Minimize the effect of background tasks on battery consumption
- Minimize metered mobile data usage, where possible

QoE-centric Policies

While modern operating systems such as Android make decisions based on policies meant to improve QoE, it is unclear that these static policies always result in the right decisions. Static policies such as the Linux ondemand CPU governor and Android’s use of cgroupgs would benefit from QoE feedback.

The ondemand CPU governor increases CPU frequency to the maximum when there is work to do. But, is this always necessary to improve QoE? If the CPUs run at a lower frequency, they run more efficiently and can improve battery life.

Android uses Linux cgroupgs to limit background tasks to ~5% CPU share. However, misuse of thread priorities or AsyncTasks can cause this policy to fail to meet its goal. Process scheduling could benefit from knowing a task’s impact on QoE.

Active Wait Detection

Active waiting, such as when a progress bar or spinner is on-screen, is a hint that the app is in a QoE-critical section. The OS can use this information to prioritize resources. We observe that active waiting produces an interesting graphical pattern, and are working on building a classifier to detect it.

Challenges in Quantifying QoE

The nature of modern apps makes automatically quantifying QoE very challenging. On Android, complicated UI hierarchies and the multi-threaded, multi-process nature of apps contribute to the difficulties in analyzing and improving QoE.

In order to measure QoE and understand a task’s contribution to it, mobile operating systems may need to be redesigned to improve their view of the system.

Quantifying QoE Via State Detection

Input events can lead to state transitions in an app’s view tree. Detecting various app states and state transitions can help the OS improve QoE. For example, the length of the state transition will often correspond to user-perceived latency, a contributing factor to QoE. By measuring the length of the state transition, we can measure user-perceived latency which will help quantify QoE.

QoE-aware Networking

Tracking network activity on smartphones from packets to pools will help mobile operating systems improve quality of experience.

Understanding a network flow’s impact on QoE will allow the OS to prioritize network resources within the device. Furthermore, understanding the flow of data from the network to the screen has the potential to reduce mobile data and battery consumption.

In order to further improve QoE for apps that use the network, the effect of a network flow on QoE should be considered by the network itself.

QoE-sensitive traffic should be prioritized over time-insensitive traffic, such as that of certain background tasks.

Improving QoE across the network will require fundamental changes. We will begin to explore protocol changes and promising technology like SDN to meet this challenge.