Measuring and Improving Smartphone QoE Using the Screen as a Sensor

Scott Haseley, Geoffrey Challen
University at Buffalo
{shaseley,challen}@buffalo.edu

1. INTRODUCTION

Smartphone users judge their apps and devices based on the performance they experience. Yet, because of the subjective and multi-dimensional nature of quality of experience (QoE), objectively measuring it is a serious challenge. Without metrics capable of capturing aspects of interactive performance, mobile operating systems cannot understand how their decisions affect QoE. Instead, we are left to design static OS policies that provide best-effort QoE, and measure overhead in terms of percentage, without understanding the effects it has on QoE.

To quantify aspects of user interaction that have an effect QoE, we propose computing QoE metrics at the point where instances of poor QoE often manifest themselves, namely the screen. Frozen apps, laggy scrolling or typing, and long user transaction latencies are examples of poor QoE that users experience through interacting with the screen. These are also examples of where the OS should understand how its resource allocation and policy decisions affect QoE.

Users experience other forms of QoE through the screen as well. For example, poorly laid out apps that are difficult to navigate might make users search through multiple screens to find what they are looking for. While the OS may not be able to improve QoE in these situations, identifying such instances can help in objectively rating apps, and can help app developers improve their apps.

To detect and improve instances of poor QoE, we propose using a smartphone's screen as a sensor. Inputs to the screen and the information it displays together contain rich information that can be used to detect instances of poor QoE. For example, by computing the screen's rate of change we can get a measure of smoothness. By determining when the screen changes state by analyzing diffs between frames, and measuring the time since the last input, we can get a measure of responsiveness. And, by exposing this information to the OS, the OS can adjust its policies and resource allocation strategies to try to improve QoE.

Our ongoing work is the design, implementation, and evaluation of QoEye, a system that uses touch input and pixel-level frame buffer output streams as input to measure and improve aspects of user QoE. This approach is appealing for several reasons. First, this view corresponds elegantly with how users experience app content. If a user's QoE is affected through interaction with the screen, then this approach should be capable of detecting it. Second, while more semantic information may be available at higher levels in the graphics stack, apps can use custom components or draw directly using OpenGL, such as the case with games and web browsers running on Android. In these cases, higher-level approaches will not work on a large subset of applications. The screen represents an ideal narrow waist that captures visual content generated by all applications regardless of how they are designed. Third, the idea is system independent, though the implementation will vary. This implies that our approach to measuring QoE is not limited to one OS, or even to mobile devices. Finally, with the increasing performance capabilities of device GPUs and additions of general purpose GPU compute libraries, the system can take advantage of the GPU for fast parallel processing over frame buffer data.

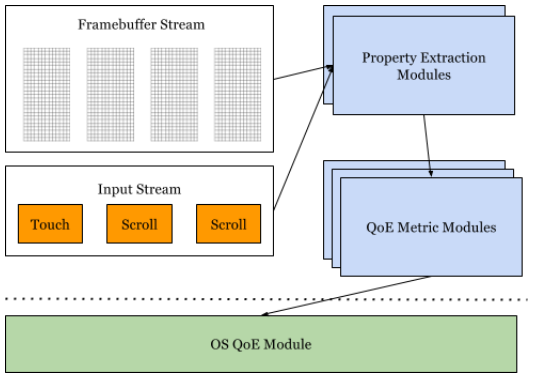

Figure 1: **The high-level system design of QoEye.**

2. SYSTEM DESIGN

The high-level design of QoEye is shown in figure 1. The framebuffer and user input event streams are captured and fed into *property extraction modules*. These modules compute high-level properties and features of the streams that can be used further downstream. For example, the modules can compute input rate statistics and frame diffs that may be necessary to compute several different QoE metrics. Next, the results of the property extraction modules are fed into *QoE metric modules*. These modules compute QoE metrics such as scrolling smoothness, latency, and responsiveness. The computed metrics are then sent to an OS-layer that evaluates and tweaks OS policies and resource decisions in order to improve QoE. For example, the OS may choose to raise or lower CPU frequencies in order to boost performance or save energy, depending on the current value of QoE metrics. This architecture provides the missing feedback that the smartphone OS needs to adjust its own actions in the interest of improving QoE.