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 metered mobile data. The management of these resources contributes to a smartphone user's quality of experience.

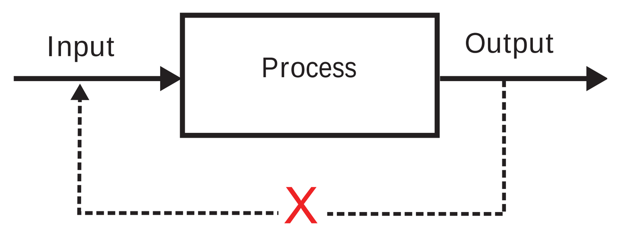
While current operating systems are adept at managing hardware resources such as CPUs, disks, and memory, there is a lot that must be redesigned to quantify and improve QoE.

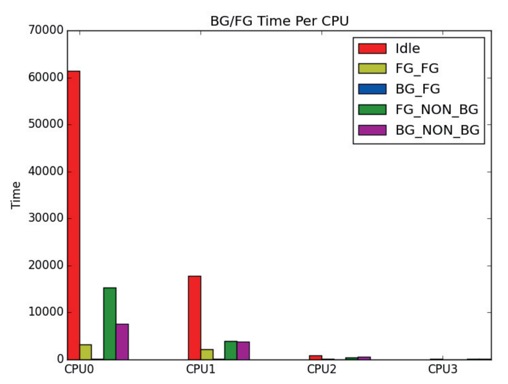
QoE-centric Design Principles

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





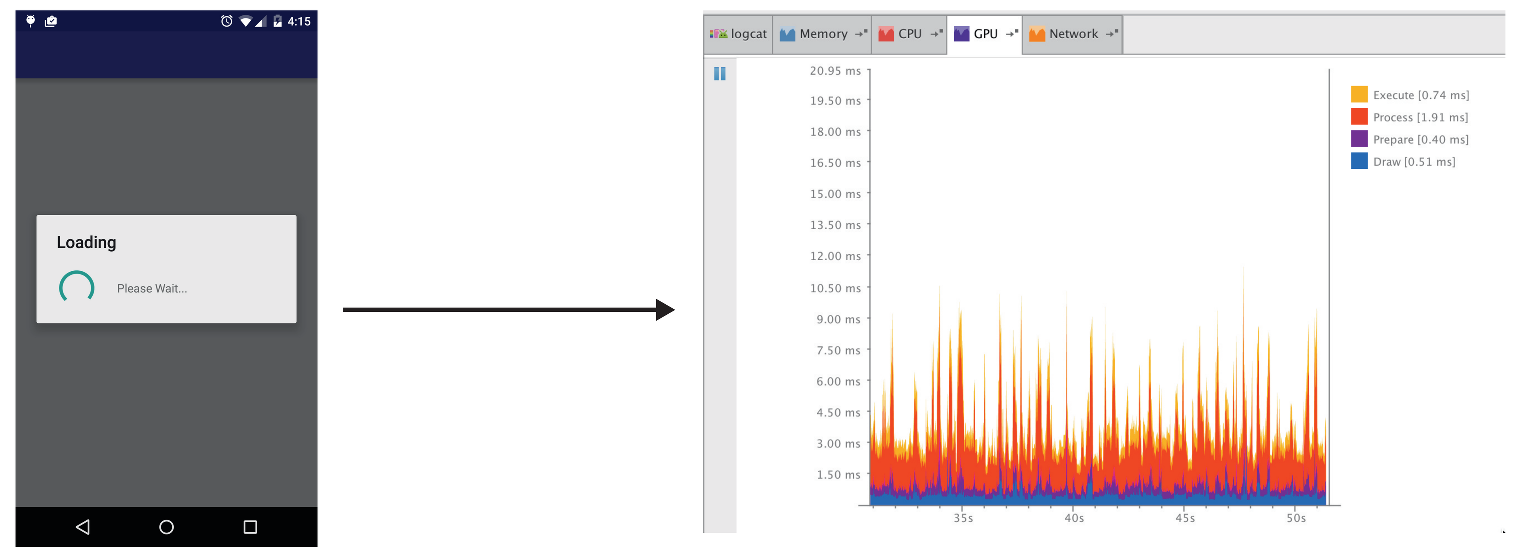
Samples from 10 devices over 7 days were taken in intervals of 1s. This plot shows intervals where both background tasks and foreground tasks ran on the same core.

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 cgroups 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.

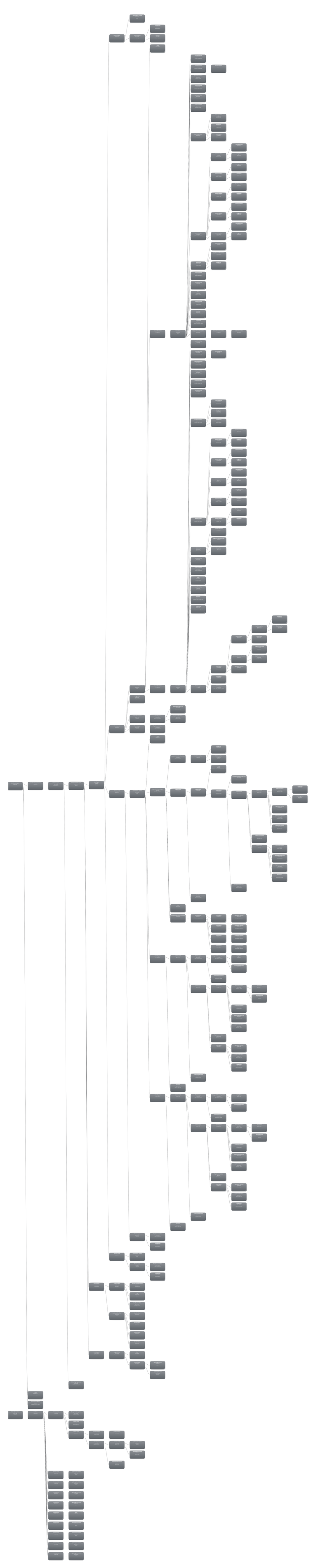
Android uses Linux cgroups to limit background tasks to ~5% CPU share. However, misuse of thread priorities or AsyncTasks can cause this policy to fail to meet its goal. Process scheduling could benefit from knowing a task's impact on QoE.

Active Wait Detection



Active waiting, such as when a progress bar or throbber 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.

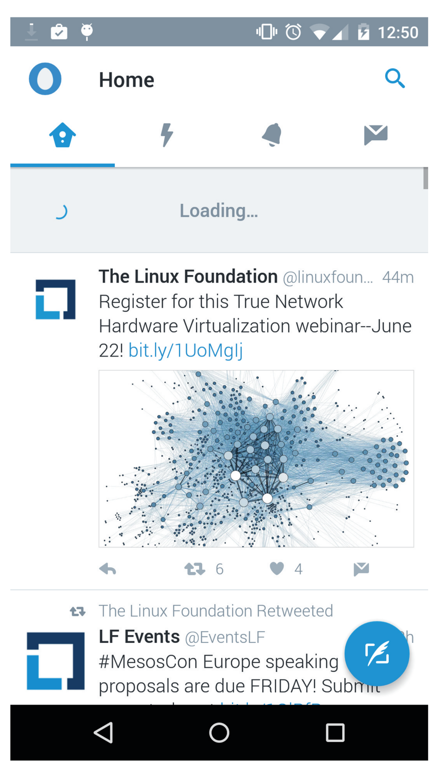
Challenges in Quantifying QoE



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.

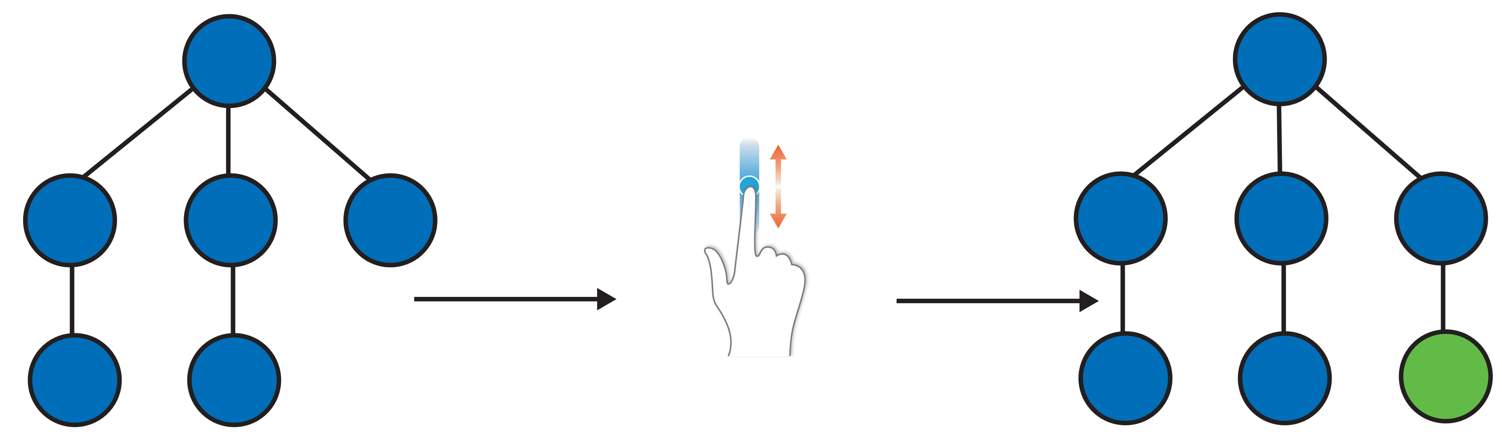
```
1526   202 com.twitter.android
1534   1526   Heap thread poo
1535   1526   twitter.androi
1536   1526   Heap thread poo
1537   1526   Heap thread poo
1538   1526   Signal Catcher
1539   1526   JDWP
1540   1526   ReferenceQueueD
1541   1526   FinalizerDaemon
1542   1526   FinalizerWatchd
1543   1526   HeapTrimmerDaem
1544   1526   GCDaemon
1545   1526   Binder_1
1546   1526   Binder_2
1553   1526   pool-1-thread-1
1557   1526   Queue
1558   1526   Queue
1559   1526   Queue
1560   1526   Queue
1561   1526   Queue
1564   1526   Answers Events
1567   1526   Crashlytics Exc
1570   1526   AsyncTask #1
1574   1526   AsyncTask #2
1577   1526   AsyncTask #3
1579   1526   AsyncTask #4
1581   1526   AsyncTask #5
1583   1526   CoordinationThr
1586   1526   q
1597   1526   Binder_3
1598   1526   pool-5-thread-1
1605   1526   GAC_Executor[0]
1613   1526   pool-5-thread-2
1616   1526   ModernAsyncTask
1617   1526   ModernAsyncTask
1618   1526   ModernAsyncTask
1619   1526   ModernAsyncTask
1620   1526   ModernAsyncTask
1622   1526   RxCachedWorkerP
1623   1526   RxComputationTh
1624   1526   RxComputationTh
1626   1526   pool-5-thread-3
1627   1526   pool-8-thread-1
1628   1526   RenderThread
1629   1526   pool-5-thread-4
1635   1526   pool-5-thread-5
1639   1526   GL updater
1640   1526   pool-8-thread-2
1641   1526   GAC_Executor[1]
1642   1526   hwuiTask1
1643   1526   hwuiTask2
1649   1526   RenderThread
1650   1526   pool-13-thread-
1652   1526   pool-7-thread-1
1653   1526   pool-7-thread-2
1659   1526   RxComputationTh
1810   1526   pool-6-thread-1
1813   1526   pool-6-thread-2
1817   1526   SoundPool
1818   1526   SoundPoolThread
1883   1526   Binder_4
2132   1526   pool-6-thread-3
2134   1526   pool-6-thread-4
2172   1526   api.twitter.com
2252   1526   ttp twitter.com
2438   1526   AudioTrack
2441   1526   AudioTrack
```



The Twitter app on Android consists of a very complicated view tree and more than 65 threads contributing to what you see on-screen!

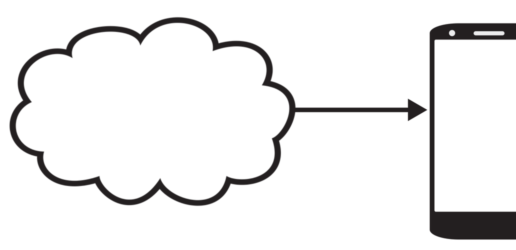
Given the operating system's vantage point, it is hard to determine what contributes to QoE.

Quantifying QoE Via State Detection



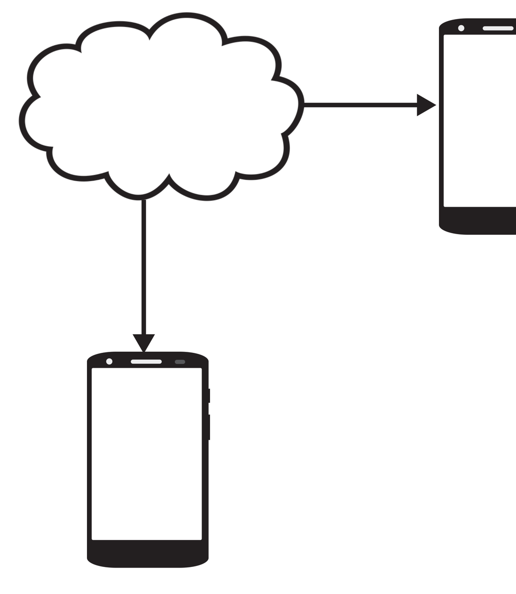
Input events can lead to state transitions in an app's view tree. Detecting various app states and state transitions can help the OS improve QoE. For example, the length of the state transition will often correspond to user-perceived latency, a contributing factor to QoE. By measuring the length of the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

QoE-aware Networking



Tracking network activity on smartphones from packets to pixels 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-sensative 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.

University at Buffalo

blue

http://blue.cse.buffalo.edu

shaseley@buffalo.edu

challen@buffalo.edu