Performance testing of the interface

Probably the most important attribute of the application is performance when showing a feed. For this reason, a lot of speed tests were made to reduce loading times and increase the overall visual flow when scrolling through a feed. This document shows some of the test data and goes through some of the methods used to improve it.

In the early tests, each item in the feed took about 60ms to create, except for the first item, which took about 135ms. This does not include the loading time for the profile picture, which took about 270ms to load.

The first thing we did to reduce time was to recycle views. The general idea of this is to take every view that gets scrolled out of screen (and therefore is no longer visible) and reuse it by basing the upcoming item on it. This way, only the number of items needed to fill the screen needs to be created, which is about 6 or 7. After some refactoring of the code, tests showed that the views being recycled took about 18ms to load. This is an improvement of 42ms per view, which improved the flow of the feed a lot.

Another implementation was the Viewholder pattern. This is used to keep the adapter from having to find all children views for each view created. The findViewById method is quite time consuming, and doing it in a list that creates views frequently while scrolling is not a good idea. In our case, each view contains 5 children views. This means 5 calls to findViewById, and scrolling in a normal pace would result in about 15 to 20 calls per second.
The implementation of this pattern quite simple. Every time a new view is being created, a Viewholder (which is an inner class) is also created, and filled with the children views. The fonts being used in them is also set. The view is then tagged with this Viewholder, and when the recycling of a view begins, that very same tag is being retrieved from the used view. This way, there is no need to call findViewById or set any fonts, since this is already done.
Tests show that this implementation reduces the time it takes to load a recycled view to about 13ms, a 5ms improvement.

The most time consuming part of the feed is without a doubt the loading of the profile pictures. This is quite obvious, since the images are being downloaded on the fly, without any caching. We chose to do this to reduce the application's load on the phone's memory. Since we can not improve our users' Internet connection speeds, there is no way of making this faster.
However, making the feed smoother does not necessarily mean having to reduce download speeds. By using an AsyncTask to download the images, a lot of time was reduced for loading multiple images, since about 5 asynchronized download calls now can be sent before the first one is finished.