Project CS 2012 - Frontend Evaluation

December 13, 2012

1 Introduction

In order to evaluate our browser application and the NetInf services, we want to find answers to the following core questions:

1. Can we reduce uplink bandwidth using NetInf?
2. How much time does it take to receive the same data through alternative mediums (Bluetooth, NRS, uplink, database)?
3. Which pages have we tested that are displayed correctly?
4. How efficient is our approach compared to a lookup in a DNS?

The test setup consists of a static set of web pages that we want to retrieve from a varying number of phones (1,2,4). The phones will access the web pages in a random order. In order to achieve a more realistic scenario, new web pages will be requested after a selected time interval. This should simulate an everyday user situation in which people actually read the content requested instead of changing the web page immediately. Note that we are not going to do research on what web pages are most popular among users, but instead use pages that we think are commonly visited in a train scenario, such as ...

1. www.sl.se,
2. www.aftonbladet.se and
3. www.ul.se.

Within this sprint we will only collect data but not evaluate yet. The evaluation itself will be done during the next sprint.

We would be grateful to receive comments from you about our ideas of evaluating the application. Please take time constraints under consideration, since the demo of this sprint will already be next Wednesday.

2 Measurement Ideas

2.1 How much Internet traffic is avoided using NetInf?

We plan to count how many times we use the 4 different methods of retrieving web pages: Via Bluetooth from another device, from the NRS, from the local database and from the internet. We also plan to combine this information with the amount of the data that was transferred to be able to show how often we can avoid using the Internet connection.

During this test we consider to vary the following variables:

1. Number of phones
2. Time intervals between accessing web pages
3. Distance between phones (for Bluetooth connectivity)
4. Publish with full-put and without full-put
5. Random number of phones that have elected to share

2.2 How much time does it take to receive the same data through alternative mediums (Bluetooth, NRS, uplink, database)?

Loading data in NetInf involves different technologies for transferring data. In our extension of OpenNetInf, we can transfer data between two devices through Bluetooth, transfer data directly from the NRS (acting as a cache node), get data from the Internet link or use the device's internal database to get previously visited data directly from the internal cache.

We want to measure the latency involved in retrieving objects using all four methods. We can use the measurements to identify potential bottlenecks in the system – for example, is the database access faster than downloading a new page? – and to validate the idea of using NetInf in scenarios such as the 'Train' scenario discussed in the project.

This test is helpful to evaluate how these different methods of retrieving objects interact with each other. Our application architecture goes through three types before using external internet connection, as it first checks the internal database, then the NRS and finally the Bluetooth neighbours. It will be interesting to see how the objects get spread after a certain amount of time through the network and how long it takes to completely load web pages with our application.

To measure such transfers, we will employ timers in each component to mark the time needed to retrieve objects. Our plan is to produce graphs with these measurements and provide a clear view of the bottlenecks and network usage for our application.

2.3 Which pages have we tested that are displayed correctly?

We want to write down a list of web pages we have tested and which were displayed correctly within our application.

This is a manual test, where we check if the web sites in our list are correctly displayed in our application. It is an interesting way to detect errors. Furthermore, developers that will need to reuse our application will have a defined set of pages they know about for sure that they are tested.