Performance Observations of Fennec Development on Mobile Networks

July 31 2008
mcmanus@ducksong.com

"What to do with all that latency?"

This memo is an interim status report, updating a similar document from 6/30/2008 which can be found here: https://bugzilla.mozilla.org/attachment.cgi?id=329249. This memo makes extensive use of the efficiency metric and several pre defined test scenarios. A description of those scenarios and an extensive discussion of the efficiency metric can be found in the 6/30 memo. All of this work is captured under bugzilla bug number 437953.

UPDATED ISSUES FROM 6/30

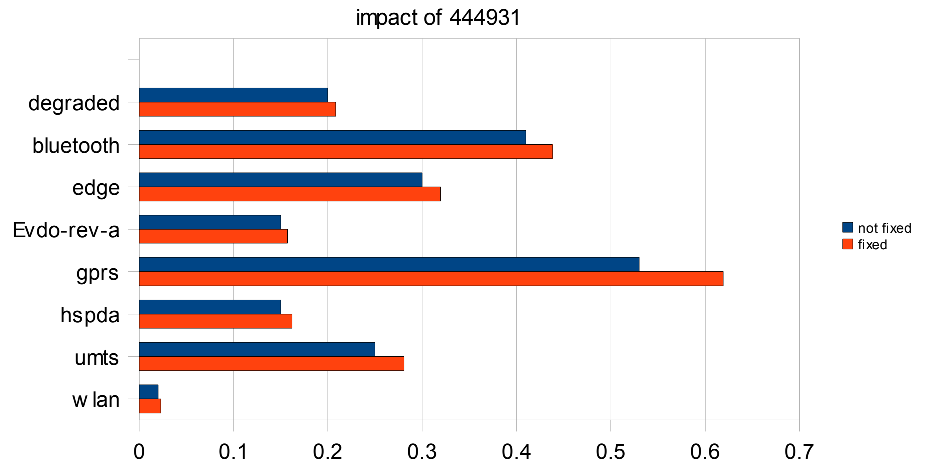
Aborted Double Fetch Problem

This issue was identified to be related to some unintentional HTML in the test case. The markup contained an img element with an empty src attribute (i.e. <img src="">). Fennec correctly interpreted that to be a relative URI reference, which in this case is the same URL as the HTML itself. Fennec went on to fetch the HTML URL again, this time sending in different HTTP Accept headers which indicated a preference for image formats. The server responded with the base HTML page again, fennec closed the connection early upon seeing a non-image content-type and the client needed to establish a new TCP connection to continue. This sequence explains the truncated connections observed in the original trace.

While this is an error in the HTML, it is both a fairly common error to observe "in the wild" and, more importantly in my opinion, a foreseeable error within the browser without ever going to the network. It can therefore be corrected without taking a 3*RTT penalty which is necessary with the current algorithm.

The bug https://bugzilla.mozilla.org/show_bug.cgi?id=444931 contains discussion of the finer points of the issue, including backwards compatibility questions, a proposed patch, and citations of 20+ other bugzilla entries which illustrate this is a commonly occurring issue.

For the tiki data set I am studying, fixing this problem yields around a 10% improvement in efficiency ratios – depending on how bad the RTT of the network being measured is.



PIPELINING EFFICIENCY

The initial work identified some pipelining in the network traces, but that it did not seem to be occurring as frequently as expected. High latency environments will require aggressive pipelining to be effectively utilized.

With further investigation I was able to characterize the problem more concisely. The HTTP pipelining code always favors opening new connections (or reusing idle ones) over pipelining onto an existing active one (which is sensible), and never pipelines a request onto an existing active connection. Instead, it bundles multiple delayed requests together when a connection becomes available and sends them all at the same time.

This results in delay being un-necessarily added to the pipelined transactions as well as the creation of contention for upstream bandwidth that could be avoided by spacing the requests out. Upstream bandwidth can be extremely limited, even in modern mobile networks.

This creates a particular problem for latency sensitive Javascript and css, which block the rendering of the base HTML page. If these files cannot be fetched in parallel with the HTML, due to lack of parallel connections permitted to the server, then they will never be pipelined because pipelining in the current implementation never takes place against the outstanding HTML transaction.

https://bugzilla.mozilla.org/show_bug.cgi?id=447866 has been filed to deal with the issue. I am fairly certain this is the correct interpretation of the code, and the impact of fixing this would be significant. I do not have a test patch for it to measure, however.

REQUEST ORDERING

The “Speculative Loader” bug https://bugzilla.mozilla.org/show_bug.cgi?id=364315 continues to play a significant role in performance for mobile environments. I have initiated a basic conversation with Blake, owner of the above bug, regarding a second generation architecture involving multiple queues for different kinds of content.

In my proposal, each queue would be tied to a network connection. The receive buffer size of each connection could be tuned in real time in order to establish higher levels of scarce network bandwidth for the higher priority queue. The obvious set of initial queues would be { html/js/css, images/media, other}.

No work other than informal design discussion has occurred.

PARALLELISM AND RELATED BEHAVIORS

The most interesting finding of the first leg of this study was that no network parallelism was observed in even a single one of the 56 data sets.

I traced this to a set of default preferences established in mobile/apps/mobile.js which explicitly limited fennec to one connection per server (up to 4 in total). Furthermore, pipelining was enabled but limited to a relatively conservative depth of two outstanding pipelined requests at any one time. (For later reference, that parameter controls the total number of outstanding pipelined requests – it is two total, not two per connection in configurations where more than one parallel connection is achieved).

The primary characteristic of the mobile network is high latency. In order to effectively utilize the network that latency must be covered up through some other technique. The two primary mechanisms for that in HTTP are pipelining and concurrency – as such the conservative values currently set must be updated.

Discussions with other development engineers agreed with this basic point, and this memo sets out to establish a data set which can be mined to identify a more optimal default configuration.

I have made 1008 distinct measurements in pursuit of this. There are 7 different data sets (pages) and each of those pages is measured on 8 different simulated networks. In turn, each of those 56 simulations was run with 18 different configurations of concurrency and pipeline depth. The test range ran from the current {1 conn, 2 depth}, up to variations with 6 concurrent connections or a maximum depth of 8. It was quickly determined that the far edges of that range were not optimal, so denser samples of depths of 2, 4, 6, and 8 were taken for concurrencies of 2, 3, 4, and 5 – but less samples were taken for concurrencies of 1 and 6.

The settings do indeed make a remarkable difference. For instance, the current default configuration came in last place (18th) 48 out of 56 times, and improved to just 17th for 5 of the 8 times it did not come in dead last.

The improvement between the best measurement and the worst measurement for any given test scenario was on average an efficiency ratio change of .26. That is a very large difference given the average ratio of the default configuration is just .45. **A average 58% improvement in network performance was achieved through tuning!**

More aggressive pipeline settings will accrue even larger benefits after the pipeline efficiency scheduling problem referenced above is addressed.

There is no single perfect setting, different tests and networks give slight preferences to different combinations. But some arrangements perform consistently well.

The best scenarios generally have a concurrency between 3 and 5. Concurrencies of 3 require very deep pipelines to perform near the top. Likewise, concurrencies of 4 with moderate pipelines perform generally as well as 5 concurrent sessions. As there is a disproportionately high memory cost of a new TCP connection vs pipelining requests onto an existing TCP session, it is prudent to use 4 connections as the base configuration and save some memory consumption.

A concurrency of 4 with a max pipeline depth of 6 was the best performer in 8 of the 56 scenarios. The only combination with more first places was a concurrency of 5 with a depth of 6, which finished first 9 times. However when looking at the number of top quartile performances {4,6} finishes in the top quartile 30 times out of 224 samples, compared to 27 for {5,6}. The individual efficiency metrics tend to be extremely close to each other.

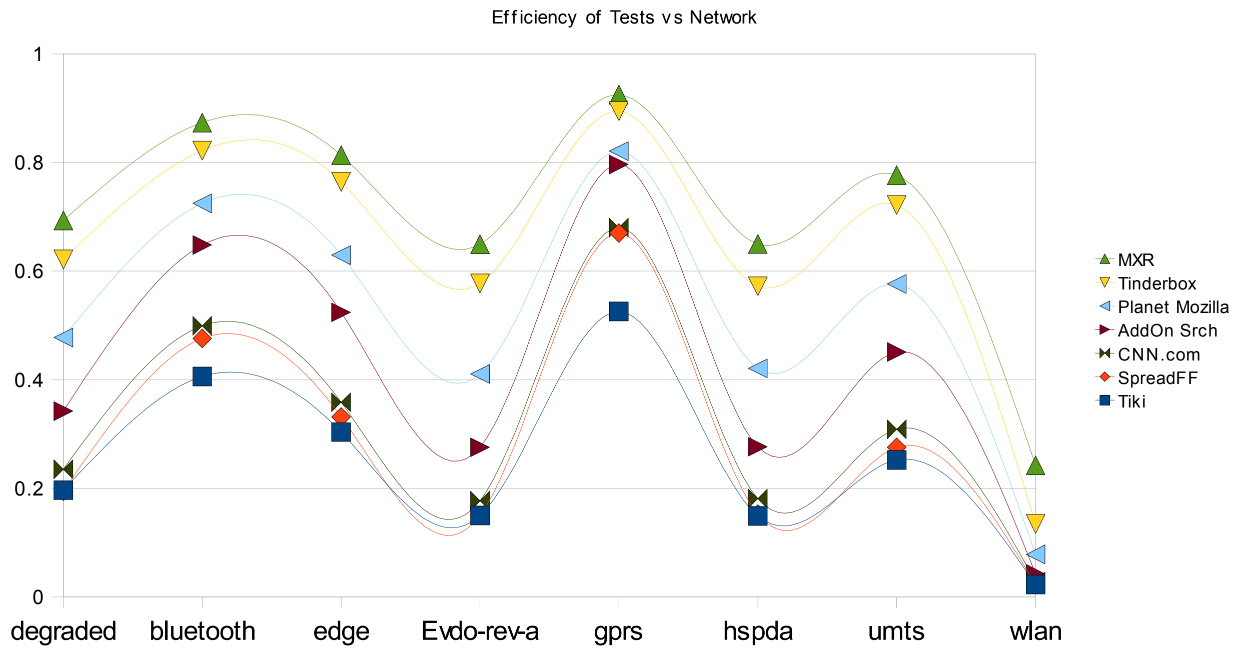
This leads to a pretty firm **conclusion that a concurrency of 4 with a pipeline depth of 6 yields consistently strong results across a wide range of pages and networks while maintaining a reasonable memory footprint.**

All of the 1008 data points will be attached to the http preference bugzilla entry: https://bugzilla.mozilla.org/show_bug.cgi?id=448805

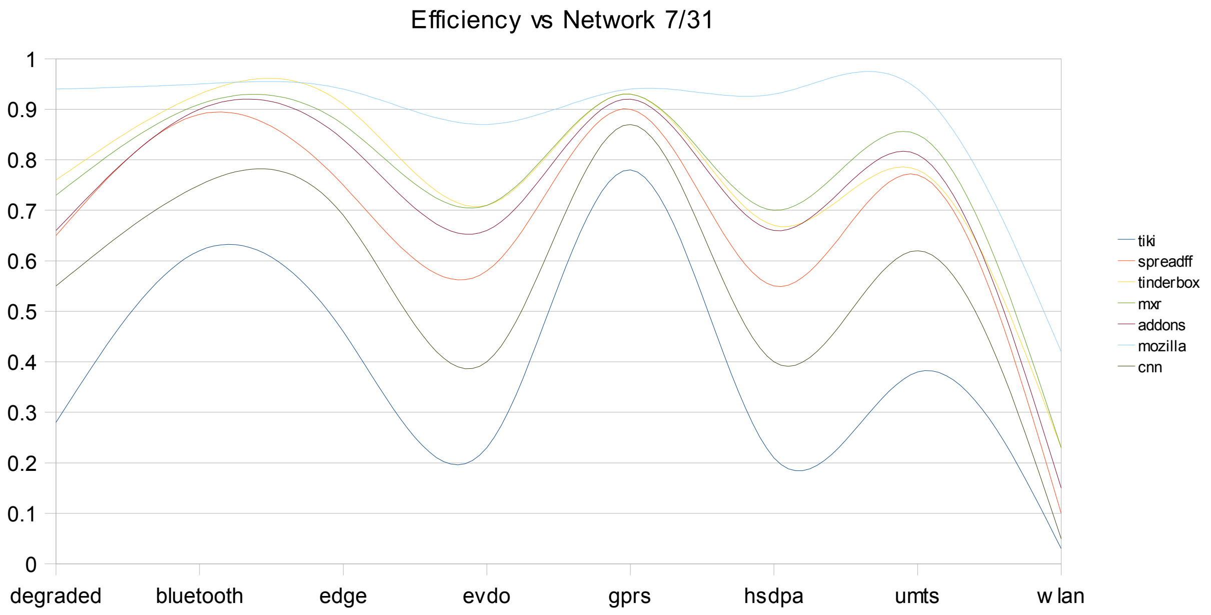
OVERALL PERFORMANCE UPDATE

It is worth comparing the performance of a build from one month ago, to that of one from today with the double fetch problem patched (with a temporary patch) and using the new configuration preferences:

This is the old graph:



And this is the current one – which shows similar relationships but significantly better absolute performance.



OTHER LESS ACTIVE WORK ITEMS STILL ON THE ROADMAP

- Integrate the moz-shaper network simulation script into talos
- Build a metric that reflects “page reading latency” instead of just load time
- Consider TCP sender congestion window bottlenecks as a source of optimization. Shared congestion control strategies may help here.
- Develop a bridge mode for the simulator – allowing it to be used with finer grained timers, and non Linux clients
- Design study of effect on intermediate network proxies and how they can be leveraged.