Performance Observations of Fennec Development on Mobile Networks

August 19 2008
mcmanus@ducksong.com

Re PIPELINING EFFICIENCY
Bugzilla 447866, part of 329249

Previously identified as an issue, was a bursty HTTP pipelining implementation.

Mozilla made its pipelining decision at the same time it obtained a fresh TCP connection. That fresh connection might be a new one, or it might be a reused persistent connection – but in either case the connection did not have any outstanding HTTP transactions on it. This new connection was filled with as many pipelined transactions that were currently queued and that the configuration supported.

However, if a transaction could not obtain a fresh connection when it was first presented to the network layer it was deferred into that queue even if one of the existing connections had room in its pipeline quota. If an existing pipelined transaction completed it could be replaced by one from the queue without waiting for the pipeline to clear completely, but it was always replaced on a one-for-one basis without regard to the configuration of maximum pipeline depth.

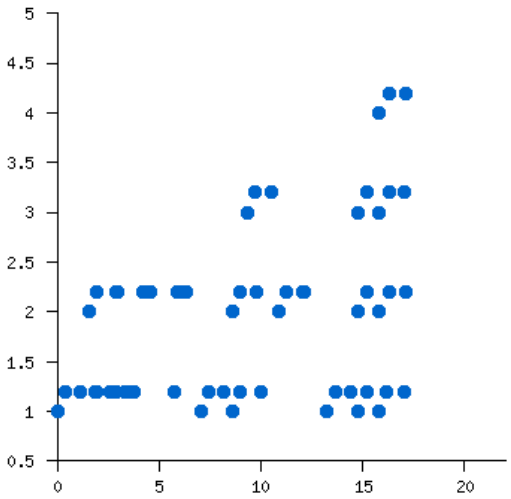
Patches are attached to the bug which more aggressively schedule already pipeline eligible transactions onto existing connections. The impact of the patches varies by test case, but on a EDGE network they tend to generate around a 10% improvement depending in page load time on the ability of the page to exhaust the pipeline depth quota. Greater improvement is seen in some cases. A reduction in packets transferred of about 3% is also typically seen.

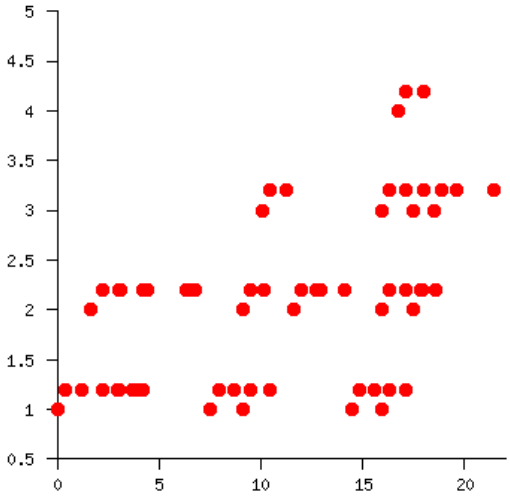
The change in behavior is best illustrated visually.

PACKET TIME SERIES

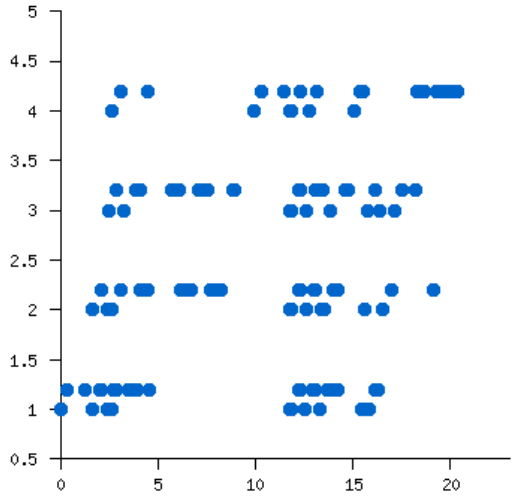
The first two graphs show a time series comprised from all the packets in the page load. The baseline fennec is shown in red, the patched version in blue. Each page load uses four TCP connections which are each easily identified as two horizontal clusters of packets. Each dot is one packet, the lower “line” in each of these are the various HTTP requests, the upper grouping are the responses. Note the blue graph shows a tighter grouping towards the left (time = 0) which is derived from lower request latency and results in better page load time.

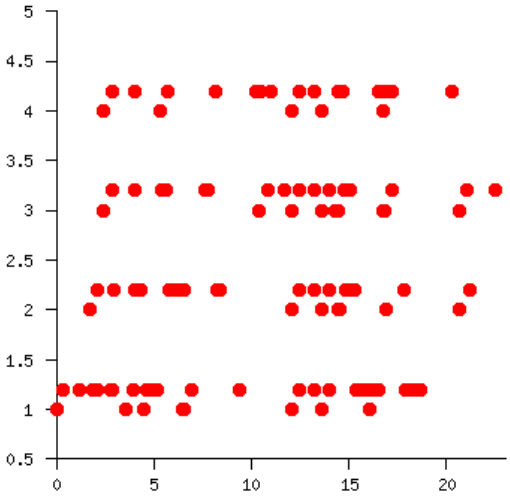
This is the “tiki” test case -





This is the “Spread Firefox” test case -

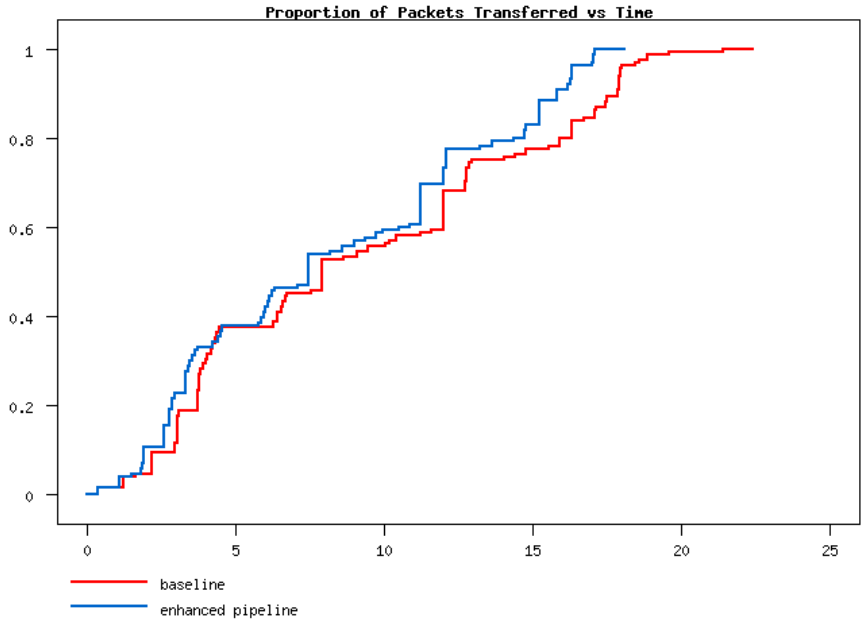




Cumulative Packets Received vs Time

Another way to visualize the data is as a proportion of all the packets in the page load transferred over time. This allows both the baseline build and the patched build to be plotted on the same graph with two lines.

This is the “tiki” Test case:



This is the “Spread Firefox” Test case: