HTTP/2 Performance

This slideshow collects data from HTTP/2 performance tests regarding latency. The tested features of HTTP/2 protocol are **multiplexing** and **server push**.

This is an assignment for Aalto University course ELEC-E7320 – Internet Protocols

Test framework

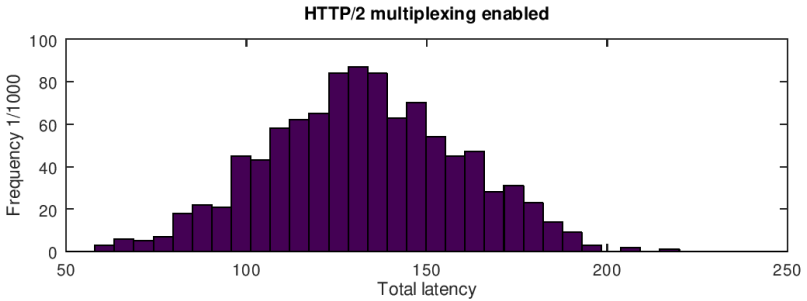
- The tests are run on a local loopback connection, with a random delay added to simulate a real-life environment (internet connection)
- Random delay (30ms + 15ms normal distributed variable) is applied by *tc(8)*
- The exact command is
  - *tc qdisc (change/add) dev lo root netem delay 30ms 15ms distribution normal*
- The tests are run 1000 times, both with the given HTTP/2 feature enabled and disabled
- **The effect to latency is only tested**, because these features (push and multiplexing) don’t really try to solve any bandwidth/throughput issues.

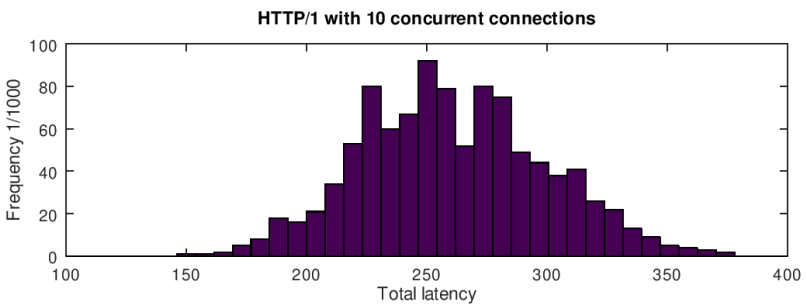
Multiplexing

- Stream multiplexing was introduced in HTTP/2. This defeats head-of-line blocking, because every request/response uses their own independent HTTP/2 stream.
- To test stream multiplexing we created two test cases (1000 iterations) with
  1) basic HTTP/2 stream multiplexing with 20 concurrent streams
  2) ~~Replicating HTTP/1.1 in HTTP/2 by creating 20 concurrent streams, but handling their responses in a chain, always waiting for the oldest stream to complete~~ (the library does not allow this, so this case was not implemented)
  3) Replicating HTTP/1 and using a limited number of connections per time (10 connections). **This is actually closest to the real-life situation**, as most of the web browsers never implemented the HTTP/1.1 pipelining.
- We used the same HTTP/2 library framework (C++14, boost, libnghttp2_asio) in both cases to minimize the internal differences between the two test cases
- The simulated delay's random variable should create some HOL-blocking in the 3rd test case

Multiplexing – test results

- The results clearly show that HTTP/2 multiplexing is superior when compared to the traditional way of opening many concurrent connections
- HTTP/2 results are really impressive, because the opening of the connection already does one RTT, and the resulting mean is about 2 * 2 * 30ms (120ms total)
- HTTP/1 is clearly slower, because of the HOL-blocking with the concurrent connections; new requests cannot be made before an old connection has closed
- We used maximum of 10 concurrent connections in this example, but many web browsers use even lower amount of them, which would make the performance here even worse
- Also a higher latency would affect the HTTP/1 harder





Histograms from 1000-iteration tests. Requests had 30ms (static) +- 15ms (normal distributed random) delay, HTTP/2 uses multiplexing, HTTP/1 uses concurrent connections

Server push

- Server push was added in HTTP/2 for faster page loading.
- Server “pushes” needed extra data before client requests for it
  - A great example is an independent stylesheet file that some HTML file needs.
- The push is done through a new push stream initiated by the server
- For this case, we created 2 test cases with

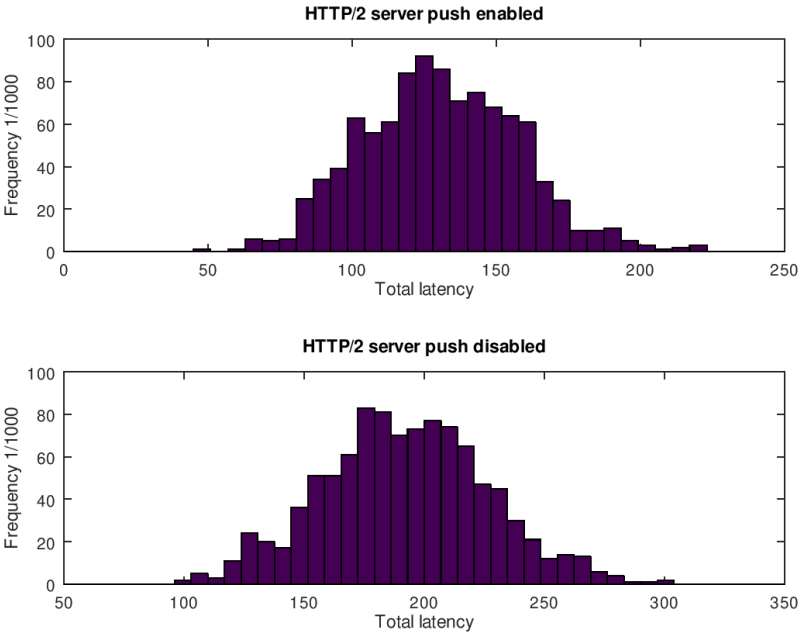
  1) HTTP/2 connection where client loads a html page containing an external stylesheet file, and server pushes this .css file

  2) HTTP/1.1 style of connection where the client loads the same html page, but requests the .css file itself

- We used the same HTTP/2 library framework (C++14, boost, libnghttp2_asio) in both cases to minimize the internal differences between the two test cases
- In this test, the added delay should show a difference in total page load time, as HTTP/1.1 needs to request the .css file separately.

Server push – test results

- As we only used one external dependency to measure the overhead caused by manually requesting the file, the difference is really self-evident;
  - With server push enabled the mean load time is 2 * RTT (120ms, 2*2*30ms). As long as all the dependencies are pushed, it most likely would not grow much even if the amo of content would grow
  - Without server push, the mean load time is about 3 * RTT (180ms). If there were chained dependencies, the load tim would most likely be (2+N) * RTT, where N is length of the longest dependency chain
- Conclusion: server push really helps with the latency, but on the other hand listing the dependencies will require extra runtime resources from the server



Histograms from 1000-iteration tests. Requests had 30ms (static) +- 15ms (normal distributed random) delay, HTTP/2 pushing enabled in upper plot, disabled in lower with one external .css file loaded.

Source code

- Source code for the complete test framework including all test cases and scripts to generate histograms with the collected test data can be found from

https://github.com/jussihi/HTTP2-perf-testing