Analysis of two browser-specific characteristics, from a performance perspective

Authors: Ragnar Lönn <ragnar@loadimpact.com> and Robin Gustafsson <robin@loadimpact.com>

1. Background

Load Impact [1] is an online load testing service, dedicated to helping people test and improve the performance of their web applications. Load Impact has developed a ***Web Page Analyzer ("the Analyzer")*** [2] that allows web developers to analyze how different web browsers perform when they load a web page. To enable the Analyzer to emulate different browsers, Load Impact has built on the work of the ***Browserscope*** project [3]. Browserscope have identified a number of performance-affecting characteristics that separate different web browsers, and these characteristics are emulated by the Load Impact Analyzer in order for the Analyzer to be able to simulate the behaviour of a real web browser in a (from a performance perspective) realistic way.

As part of the further development of the Analyzer, Load Impact has studied a couple of newly identified browser-specific characteristics, their performance impact, and how to test them (how to test if a browser exhibits a certain characteristic). This report aims to explain the findings from the study.

2. Browser-specific characteristics studied

2.1. Support for HTTP trailing headers

Commonly, an HTTP transaction starts with a client making a request to the server, asking for some resource (file/object). If the resource exists, and the server is able to return it, the server will give the client a positive response (HTTP return code 200), then send a number of HTTP headers that contain metadata information about the object that is to be transferred, such as what type of object it is (is it an image? a text file?), how big it is (how many bytes), etc. When the HTTP headers have been transferred, the actual object will be sent. This is called the "body" of the HTTP message, the actual data that the client asked for.

However, in HTTP 1.1 [4] there is a new feature available when using the chunked transfer-encoding [5], that allows a server to send extra HTTP headers **after** the body of the message has been sent. This is to facilitate sending of metadata information that might not be known at the time the body is being transferred, or where the performance of the transaction can be improved by delaying the sending of header data, allowing the client to get the actual content it wants quicker. An example of where this is useful is if you have a system that generates the response data dynamically and then wants to include a checksum HTTP header, Content-MD5 [6], that contains a checksum of the data sent. The server does not know the checksum until the dynamic object has been fully generated, so unless it can send the checksum header after the body of the message, it has to generate all the data first, before sending a single byte. This can mean a substantial delay for the user, if there is a lot of data to generate and it takes a while, and it will also mean the server has to buffer all the data locally before sending it, which can be memory-consuming and result in significantly higher server costs to operate the service.

Not all browsers (quite few in fact) support HTTP trailing headers, which means it is an interesting characteristic that differentiates browsers and that can affect performance.

2.2. Ajax HTTP POST requests are sent as two or more TCP packets

This is an obscure phenomenon that is most likely an unintended side effect of how most modern web browsers are written. A browser capable of executing Javascript [7] (most browsers) can generate HTTP requests from within the Javascript code. This is what services like Google Gmail [8] are built upon, it is the foundation of the Ajax technology [9].

The Javascript code can generate both GET and POST (almost any type of HTTP request actually) requests and, in the case of GET requests, any tiny request will always be sent over the network in a single TCP packet, resulting in a data transfer that is optimally efficient. However, as it turns out, most browsers will split an HTTP POST request into two separate TCP packets, with one packet containing the HTTP headers, and one packet containing the HTTP body. The browsers will only do this for HTTP POST requests generated by Javascript, however. It is a quite obscure phenomena, but nevertheless something that could have a performance impact for many modern services that are Javascript-heavy.

A majority of browsers seem to exhibit this behaviour (sending Ajax HTTP POST requests as two or more packets) [10] and [11]

3. Detecting the characteristics

3.1. How to detect if a browser has support for trailing HTTP headers

To detect browser support for trailing headers we augmented the Browserscope project with a new, network category, test. The test consists of a single HTML [12] document with a Javascript resource loaded by means of a <script> tag. The Javascript resource is generated dynamically by a CGI script [13] on the server side and returned in a chunk transfer-encoded [14] HTTP response with a Set-Cookie HTTP header [15] set in the trailer.

It is necessary for the Javascript resource to be located on the same domain, or a subdomain, as that of the HTML document for the cookie to be treated as a first-party cookie and not be discarded as a third-party cookie which is the default behavior in some browsers, notably Internet Explorer without the use of a P3P [16] compact policy [17].

We attach a Javascript event handler to the window.onload [18],[19] event to be notified once the document and all its referenced resources have been fully loaded. Once notified, we dynamically add a new <script> tag to the DOM [20] to load a second Javascript resource from the same domain as the first Javascript resource. It is essential to load the two Javascripts from the same domain to make sure the cookie is sent back in the request for the second Javascript resource.

The second Javascript resource is dynamically generated server-side by the same CGI script as the first Javascript resource, albeit with a different query-string. This time the CGI script checks if the cookie is present in the HTTP request headers. If the browser picked up the cookie sent in the trailer of the HTTP response for the first Javascript resource it should send the cookie back to the server in the request for the second Javascript resource and thus manifest its support for sending headers in the trailer of a chunk transfer-encoded HTTP response.

The code of the second Javascript resource calls a Javascript function present in the document which inserts a text string into the document to let the viewer know if the browser has support or not.

3.2. How to detect if a browser sends Ajax HTTP POST requests as two or more TCP packets

To detect this behavior we constructed a simple HTML document with a single button which upon being pressed generated an XHR (XMLHTTPRequest) [21] POST request with a small amount of POST data. Using Wireshark [22] we observed the TCP packets sent as a result of the XHR POST request for a range of different browsers and operating systems. Same technique as that used by Joseph Scott [11].

4. Current status for most common browsers

4.1. Testing methodology

All testing was performed with bare browser installations with no plugins or extensions installed. Every test was performed at least 3 times, with a cleared cache in between, to make sure the observed results were accurate.

4.2. What browsers support trailing HTTP headers

These are the results from running the trailer test we developed for Browserscope:

Windows XP

Browser	Supports headers sent in trailer
Avant 11.7	No

Chrome 4.1.249.1045	No
Firefox 1.0.8	No
Firefox 3.6	No
Flock 2.5.6	No
Internet Explorer 5.5	No
Internet Explorer 6	No
Internet Explorer 7	No
Internet Explorer 8	No
Opera 9.20	Yes
Opera 10.10	Yes
Safari 3.1	No
SeaMonkey 2.0.3	No

Windows 7

Browser	Supports headers sent in trailer
Internet Explorer 8	No
Firefox 2.0.0.20	No
Safari 4.0.5	No

Ubuntu Linux 9.10 Karmic Koala

Browser	Supports headers sent in trailer
Chrome 5.0.356.0 beta	No
Chrome 5.0.371.0 beta	No
Epiphany 2.28.0	No
Firefox 2.0.0.14	No

Firefox 3.5.8	No
Galeon 2.0.7	No
Konqueror 4.3.2	No
Opera 10.01	Yes

4.3. What browsers send Ajax HTTP POST requests as two or more TCP packets

These are the results from generating XHR POST requests and observing TCP packets sent using Wireshark:

Windows XP

Browser	Sends XHR POST requests as 2 or more packets
Avant 11.7	Yes, 2 packets
Chrome 2.0.172.43	Yes, 2 packets
Chrome 4.1.249.1045	Yes, 2 packets
Chrome 5.0.342.9 beta	Yes, 2 packets
Firefox 1.0.8	No, 1 packet
Firefox 3.0.13	No, 1 packet
Firefox 3.5.2	No, 1 packet
Firefox 3.6.3	No, 1 packet
Flock 2.5.6	No, 1 packet
Internet Explorer 5.5	Yes, 2 packets
Internet Explorer 6	Yes, 2 packets
Internet Explorer 7	Yes, 2 packets

Internet Explorer 8	Yes, 2 packets
Opera 9.20	Yes, 2 packets
Opera 9.27	Yes, 2 packets
Opera 10.10	Yes, 2 packets
Safari 3.1	Yes, 2 packets
Safari 4.0.3	Yes, 2 packets
SeaMonkey 2.0.3	No, 1 packet

Windows 7

Browser	Sends XHR POST requests as 2 or more packets
Internet Explorer 8	Yes, 2 packets
Firefox 2.0.0.20	Yes, 2 packets
Safari 4.0.5	Yes, 2 packets

Ubuntu Linux 9.10 Karmic Koala

Browser	Sends XHR POST requests as 2 or more packets
Chrome 5.0.371.0 beta	Yes, 2 packets
Epiphany 2.28.0	Yes, 2 packets
Firefox 2.0.0.14	Yes, 2 packets
Firefox 3.5.9	No, 1 packet
Galeon 2.0.7	No, 1 packet
Konqueror 4.3.2	Yes, 2 packets
Opera 10.01	Yes, 2 packets

MacOS X 10.5.8 (Intel)

Browser	Sends XHR POST requests as 2 or more packets
Chrome 5.0.342.9 beta	Yes, 2 packets
Firefox 3.6.3	No, 1 packet
Safari 4.0.5	Yes, 2 packets

It is interesting to note that Firefox 2 on the tested platforms deviates from other versions of Firefox by sending 2 TCP packets. Konqueror also deviated by sending all headers except one, Content-Length, in the first TCP packet and then the Content-Length header along with the POST body in the second TCP packet.

5. Performance implications

5.1. Performance implications of missing support for trailing HTTP headers

5.1.1. Discussion

There are several use cases where it would be beneficial, from a performance perspective, to send the HTTP response body as fast as possible, deferring many of the HTTP headers to the trailer. Three such use cases are:

- A system generating response data dynamically wanting to include a Content-MD5 checksum header to enable integrity checking on the client-side.

- A system using atleast one special purpose HTTP extension header, where the time it takes to produce the header's value is significant, but the client doesn't need the header information in order to start processing the data.
- A system where client-side user experience is tightly coupled to data arrival rate, think online collaboration tools and browser MMO [23] games.

The lack of browser support for receiving headers sent in the trailer can impact performance negatively on both client- and server-side. On the server-side the negative effect may manifest itself in the excessive usage of memory needed to buffer the entire response body in memory as compared to the amount of memory needed if the response body would be streamed. The client-side may suffer from higher user perceived latency as the response body is kept longer on the server-side, headers having to be sent first, before being sent off to the client.

To further illustrate the implication on the server-side we can imagine a web application serving dynamically generated PDF [24] files to clients. Along with each PDF comes an MD5 checksum set in the Content-MD5 header. Each generated PDF is roughly 10 MiB in size and the web application needs to handle 1000 concurrent clients at peak load. Most browsers do not support headers being sent in the trailer, with the notable exception of the tested versions of Opera, thus the web application has to buffer every PDF in memory, in its entirety, to allow the MD5 checksum to be calculated and sent in the HTTP response headers before the generated PDF can be sent as the HTTP response body. This means that the web application, at peak load, would need roughly 10 GiB of memory just for buffering. If the web application could instead send the Content-MD5 header in the trailer the server would need a very small amount of memory for buffering. The PDF would then be streamed to the client while being generated and the MD5 checksum calculated in an iterative fashion and sent as a header in the trailer, after the transmission of the body.

5.2. Performance implications of sending Ajax HTTP POST requests as two or more TCP packets

5.2.1. Discussion

Transmitting extra, unnecessary packets is always wasteful. Transmitting two packets instead of one has several negative consequences:

- It increases network traffic as every packet sent carries an overhead in the form of packet headers
- It doubles the risk of one packet getting lost in transit, which would result in a retransmit after a timeout (using up more bandwidth and delaying the whole transaction)
- It uses up more server and network router resources, as twice the number of packets have to be handled by both endpoints and network intermediaries

The primary question we wanted to answer was therefore not whether it was worse to send two packets instead of one. The question was instead how large the negative performance impact is, when you send two packets. Is it insignificant, or would it make sense to "fix" the browsers that send two packets, so they start sending one instead?

This is of course not an easy question to answer. The performance impact of this browser behaviour depends a lot on what type of web service we are looking at. One web service might use Ajax HTTP POST requests very infrequently, or not at all, while another web service might have clients that use them thousands of times per day. Note also that the biggest difference you get, is when you have an application that sends a lot of very small Ajax POST requests - small enough to fit in one TCP packet. That means e.g. Safari users will send twice the amount of TCP packets that Firefox users do. It would seem logical, that for a web service that makes frequent use of such Ajax HTTP POST requests, and that has users connecting via high-latency, high-packetloss links (such as satellite or other long-distance connections), this behaviour could indeed have a noticeable impact on performance, at least from the perspective of the user.

5.2.2. Experiment

To test the possible performance impact of bad network conditions, combined with Ajax HTTP POST requests being sent as two separate TCP packets, we decided to create an experiment.

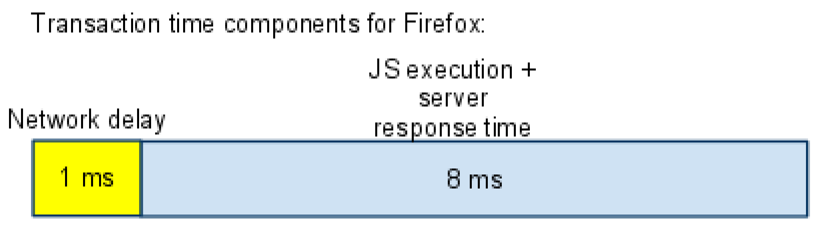
We set up an Apache [25] web server on a Linux [26] machine, and let it deliver a page that contained a Javascript. Upon execution, the Javascript performed 3000 HTTP POST requests in sequence, as fast as possible but with a 200 ms sleep between each request, to the web server. The transaction times for each transaction were computed by the Javascript, and an average transaction time was calculated.

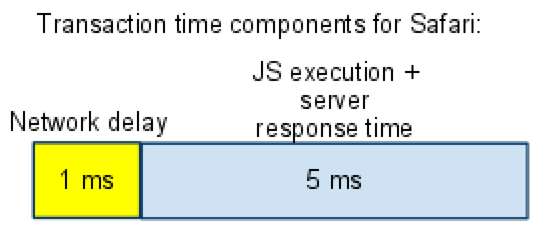
Note also that because the requests were made in quick succession, the TCP connection between client and server tended to stay open during the whole of each 3000-transaction session. That means there were only a single TCP handshake/connection setup initially in each test run.

We ran this test first in Safari 4.0.5 - a browser that we know sends Ajax HTTP POST requests as two separate TCP packets. Then we ran the test in Firefox 3.6.3, which is one of few browsers that send the same type of request as one single TCP packet. Then we ran the test in Safari again, then in Firefox, and so on until we had a total of three runs for each browser. The average transaction response times were recorded on all occasions, and these were the results:

Run	Firefox average transaction time (ms)	Safari average transaction time (ms)
1	11	6
2	9	5
3	8	6
Average	9	6

As you can see, the average transaction time was actually lower in Safari, which sends two TCP packets for each request. A likely cause is that the Javascript engine in Safari is quite a bit faster than in Firefox [27], and as we have practically no packet loss in the network, the time penalty for sending two packets is likely to be insignificant. The network delay (roundtrip delay) averaged approx. 1 ms, as measured by the ping [28] utility at several points in time.


Transaction time components for Firefox:
Network delay
JS execution + server response time
1 ms
8 ms


Transaction time components for Safari:
Network delay
JS execution + server response time
1 ms
5 ms

Then we used the ipfw [29] utility to simulate packet loss on the web client ("ipfw pipe 1 config plr 0.1; ipfw pipe 2 config plr 0.1; ipfw add 1 pipe 1 ip from any to webserver-ip; ipfw add 2 pipe 2 ip from webserver-ip to any). We set the packet loss rate to 20% (10% loss of incoming packets and 10% loss of outgoing packets) and reran the tests using Safari and Firefox (3 times 3000 requests for each browser). The results were as follows:

Run	Firefox average transaction time (ms)	Safari average transaction time (ms)
1	92	57
2	95	55
3	91	57
Average	93	56

Notable here is that the transaction time for Firefox actually increased more (approx. +84 ms) than the transaction time for Safari (approx. +50 ms). If we look at percentage increase, Firefox transaction time increased by +930% and Safari's by +830%.

The average transaction time should be a function of the following basic variables:

- Javascript execution time
- Server response time
- Network RTT (roundtrip time / delay)
- Packet loss
- Number of packets sent

Normally, when there is no packet loss, the time it takes to perform a transaction should just be the javascript execution time, plus the network RTT, plus the server response time. When there is packet loss, however, extra delay is added for every lost packet.

When a packet is lost, the operating system's TCP implementation waits a certain multiple of the network RTT [30] for an acknowledgement from the remote host, before considering the packet lost and retransmitting it. This means that every time we lose a packet we incur quite a big delay to the transaction that is taking place. If a retransmitted packet should happen to get lost, the delay will be exceptional as the TCP algorithm increases the retransmission timeout (RTO) value exponentially each time it has to resend the same packet.

To test how much of the extra delay was dependent on network delay (RTT), we did another round of tests where we added an extra 200 ms of network delay.

To establish a baseline, we first ran a test where we **only** simulated high network delay, but no packet loss. We set network delay to 200 ms (100 ms in each direction). These were the results:

Run	Firefox average response time (ms)	Safari average response time (ms)
1	212	208
2	212	207
3	211	208
Average	212	208

Then we added the 20% packet loss emulation again (10% in each direction, along with 100ms delay in each direction). Now we got these results:

Run	Firefox average response time (ms)	Safari average response time (ms)
1	324	297
2	327	296
3	323	303
Average	325	299

This was unexpected. Safari, sending two TCP packets for each POST operation, is still getting better transaction times than Firefox, which is sending one TCP packet for each POST operation. We decided to see if packet loss in one direction only made a difference, so we set the network emulator to drop 20% of all packets going **from client to server**. We left the delay settings the same as earlier, with 100 ms delay in each direction.

Run	Firefox average response time (ms)	Safari average response time (ms)
1	320	271
2	316	273
3	309	290
Average	315	278

The result was that Safari was doing even better. So, we decided to make a final test where the packet loss was 20% **from server to client** and see what that would result in. Here are the figures:

Run	Firefox average response time (ms)	Safari average response time (ms)
1	359	365
2	362	347
3	355	342
Average	359	351

Here, finally, it seems that Safari might be starting to "catch up" with Firefox, in a negative way. I.e. the adverse network conditions seem to be affecting Safari about as badly as they do Firefox.

5.2.3. Conclusion

These results were a bit unexpected. We had expected Safari to do a lot worse than Firefox in a packet-loss environment, as sending two packets doubles the chances of a packet getting lost. The outcome, however, was that packet loss seems to affect both browsers Ajax POST transactions about the same, with Safari actually doing slightly better in most cases. Only when packets are lost in the direction server->client does it seem that both 1-packet and 2-packet HTTP transactions suffer equally. We are going to investigate this a bit further, and see if it could be e.g. that in the 2-packet case, the TCP implementation gets some advantages when sending more TCP packets - it could help it maintain a better knowledge of actual network roundtrip times, and it might benefit some from a lower retransmit timeout (RTO) when the first packet in a POST transaction is lost, but the second packet gets ACKed quickly by the receiver.

At any rate, the initial assumption - that sending more packets is always bad - has proven wrong. Sending 10 packets might be worse than sending 5 (though based on our current track record in making predictions, we should probably not make such a guess either...), and there is no doubt that bandwidth usage increases, but for transaction performance when there is no bandwidth shortage, sending 2 packets might actually be better than sending one.

Appendix I: References

[1] http://loadimpact.com
[2] http://loadimpact.com/pageanalyzer.php
[3] http://browserscope.org
[4] http://tools.ietf.org/html/rfc2068, http://tools.ietf.org/html/rfc2616
[5] http://tools.ietf.org/html/rfc2616.html#section-3.6.1
[6] http://tools.ietf.org/html/rfc1864.html
[7] http://en.wikipedia.org/wiki/JavaScript
[8] http://gmail.com/
[9] http://en.wikipedia.org/wiki/Ajax_(programming)
[10] http://developer.yahoo.com/performance/rules.html#ajax_get
[11] http://josephscott.org/archives/2009/08/xmlhttprequest-xhr-uses-multiple-packets-for-http-post/
[12] http://en.wikipedia.org/wiki/HTML
[13] http://tools.ietf.org/html/rfc3875.html
[14] http://tools.ietf.org/html/rfc2616.html#section-3.6.1
[15] http://tools.ietf.org/html/rfc2109
[16] http://www.w3.org/TR/P3P/

[17] http://support.microsoft.com/?scid=kb;en-us;260971

[18] http://www.w3.org/TR/REC-html40/interact/scripts.html#h-18.2.3

[19] http://www.w3.org/TR/Window/

[20] http://www.w3.org/DOM/

[21] http://www.w3.org/TR/XMLHttpRequest/

[22] http://www.wireshark.org/

[23] http://en.wikipedia.org/wiki/Massively_multiplayer_online_game

[24] http://en.wikipedia.org/wiki/Pdf

[25] http://httpd.apache.org/

[26] http://www.linux.org/

[27] http://blog.blenderheadstudios.com/web-design/comparing-browser-javascript-execution-speed/

[28] http://en.wikipedia.org/wiki/Ping

[29] http://en.wikipedia.org/wiki/Ipfirewall

[30] http://www.rfc-editor.org/rfc/rfc2988.txt