Position Paper - Alternatives to traditional latency measurement techniques

Version 01 June 2014

LACNIC Labs labs.lacnic.net
labs@lacnic.net

Abstract

Latency's increasing importance encourages research on how to push down Round Trip Times (RTTs) across the Internet, and develop new measurement techniques. In this scope, we present a measuring alternative which involves determining TCP latency from a JavaScript tool. This alternative has the benefit of generating massive amounts of data, from probes located in many different places and thus reflecting the true state of a network, but the disadvantage of potentially introducing noise to the measurements. The benefits and problems tackled are addressed throughout this paper, as well as some potential fields of application of this new measuring alternative.

Introduction

Latency is a critical network parameter that affects services across the Internet in multiple ways. In order to have an understanding of a network's quality capabilities, or to determine the network's reachability/interconnection, latency has to be monitored constantly, from many points, to many other points. Usually, monitoring tools are located in few nodes and generate measurements to many other nodes, in a sort of *star diagram*. In order to drive more complete conclusions, there is room to develop a tool that runs in many nodes, and generate measurements to many other nodes, thus getting a better picture of the network's latency, in a sort of *mesh diagram* or matrix.

Latency is a broad term, varying with OSI stack layer, protocol, OS, hardware, network architecture, routers, network congestion and many other parameters. This paper narrows the term *latency* to *TCP latency*, measured from a web browser via JavaScript, to many web servers on TCP port 80, across the Internet. Uncertainty has to be addressed and modeled correctly, in order to minimize errors and statistical noise, and therefore generate veridic results.

TCP latency and its increasing relevance

There is a general feeling that latency measurements have been shadowed by throughput measurements, despite the close relationship that binds latency and throughput. Lets imagine a simple and every-day HTTP transfer: before any HTTP-based process begins, a TCP transaction has to be started. Both TCP and HTTP protocols require several packets to travel over the net, adding up RTTs with each packet and making the final transfer time increase. For our simple HTTP transaction, we are talking about 1,5 RTTs before the server receives the corresponding resource request (GET). It is easy to see that if latency is reduced, the global transaction time is reduced, and therefore network throughput is increased. This well-known fact has put latency in a whole new light, and new ways to reduce latency drastically are now subject of research.

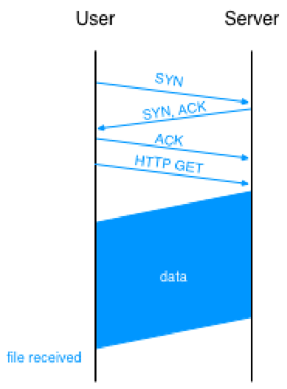

Figure 1 - HTTP File transfer [1].

Latency delays have implications in many areas, including general web surfing. According to a survey conducted by Internet Society in 2012 (ISOC's Global Inernet Survey 2012), the top reason holding general Internet users up from increasing their Internet usage was connection speed. Assuming a large portion of Internet traffic is HTTP-based, latency enhancements would impact in a considerable way among society.

This facts set an ecosystem to determine a latency baseline indicating its performance throughout a network. Having such baseline, subsequent latency improvements would be compared against historical data, and decision-making information could be drawn. This baseline could indicate whether an infrastructure investment had a positive impact or not, and in which amount. It could set metrics for content providers, in order to promote fast and responsive local content over remote content prone to latency delays. Latency within an Autonomous System could be monitored, or even further, the "distance" between Autonomous Systems could be determined and any decisions regarding traffic exchange could be made, based solely on latency metrics.

This simple scenario encourages us to find a way to measure latency easily and massively across multiple points of the net. These points, or probes, would submit their measurements to a central database where historical data would be stored and analyzed.

The instrument

Latency can be measured in a variety of ways, depending on the type of application, OSI layer, protocol, among other configurations. The approach chosen this time is to imagine the network as a black box, and generate large enough amounts of data in order to consider the measurements representative of the network's latency.

In order to measure TCP latency in an accurate way, the sampled population had to be as large, representative, and random as possible. This could only be done with a measuring instrument that follows some rules:

1. It must use technologies that result in large amounts of data generated.
2. In order to represent the world as it is, the instrument needed to be triggered from many different locations as possible, within the region of study.

One low-cost and quick-deploy option is to use massive services already in the web and web standards, like already-there web servers and JavaScript. A web page visitor triggering background latency measurements à la Google Analytics will clearly generate a large and pseudo-random data set.[1]

The following snippet shows a simple way of measuring TCP latency from a web browser in JavaScript.

```
function latencyTest(ip) {

        var ts, rtt, url;
        var to = 1000;

        if (ip.version == '4') url = "http://" +ip + "/" + Math.random();
        if (ip.version == '6') url = "http://[" +ip + "]/" + Math.random();


        $.jsonp({
                type : 'GET',
                url : url,
                dataType : 'jsonp',
                timeout : to,
                error : function(xhr, textStatus, errorThrown) {
                        if (textStatus != 'timeout') {
                                // If there is an error and the site is online, we can
assume it is due to 404
                                rtt = (+new Date - ts);
                                post(rtt);
                        }
                }
        });

        ts = +new Date;
}
```

On the one hand, JavaScript enables measurements to be massive and diverse. The test runs on the client side (web browser), and allows to be run from multiple locations (ASs). But on the other hand *the data* set gets polluted as a consequence of many known and unknown factors, such as making measurements at OSI layer 5, and the differences that may arise between browser's JavaScript engine implementation (JavaScriptCore, V8), among many others. This problems are not minor and will be worked around in the next sections.

HTTP GET and TCP latency

What is the snippet above measuring? Is it measuring actual TCP RTT? As it is, the snippet is not measuring TCP RTT alone.

Is the HTTP connection establishment process shown in Figure 1 consistent across devices? Surely not. Let's examine it again:

[1] It could be argued that the users accessing a certain web site do not vary much and end up biasing the data generated. While this statement is true, it could also be argued that by establishing a group of sites offering latency measurements, different audiences are reached and thus the measured data ends up being representative.

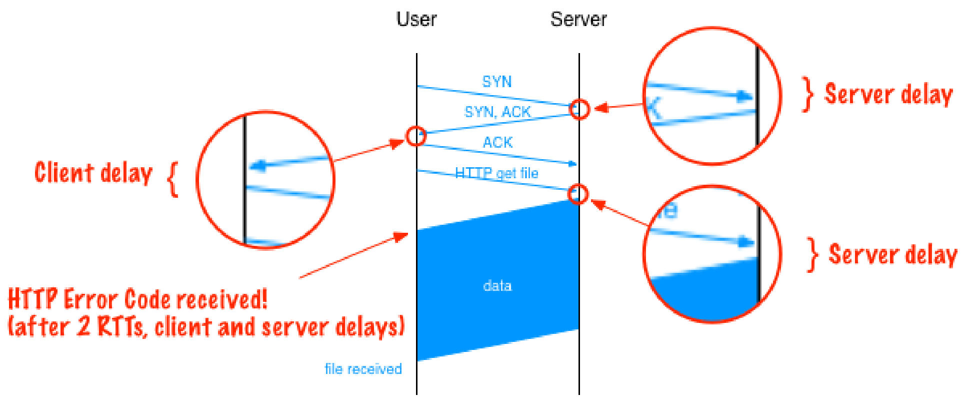

Figure 2 - Error sources implicit in HTTP-based TCP latency measurements.

Let's consider the following:

1. Both client and server introducing delays. Server delay: does it depend on server load? Server OS? Web server application?
2. The script relies heavily on hitting a 404 Return Status. Heavy 404 pages may rise an issue, as the TCP payload gets bigger so do the latency measurements (stopwatch stops at *file received* moment in Figure 2). Probably doing a GET to an unavailable port instead of an unexistent makes the script 404-independent? Would that be faster? It could be an option, or even another measurement tool...
3. Additional error: the client browser's JavaScript engine implementation (covered later).
4. Actually 2 RTTs (after the TCP 3-way handshake) occurred before the HTTP Headers arrived! One simple approach:

```
...

rtt = (+new Date - ts);
rtt = rtt / 2;

...
```

Most of the tests will be performed against different destinations, and therefore the measurements will occur during TCP connection establishment, but most of them will be performed more than once against the same destination, where a connection has already been established. The latter tests won't open a new TCP connection, but rather use the already-open TCP connection in order to maintain a persistent HTTP connection. The time required for the already-open connection phase happens to be half of the time required for the TCP connection establishment phase. Let's examine it:

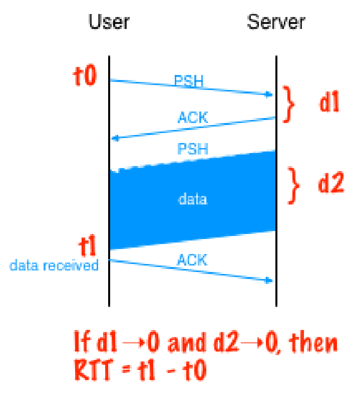

Figure 3 - Persistent TCP connection anatomy.

At t1, and considering light 404 pages as well as not loaded servers, the HTTP Error Code arrives after about 1 RTT. Considering the above example where the HTTP Error code arrived after 2 RTTs, we should consider:

```
// 1st GET
rtt = (+new Date - ts);
rtt = rtt / 2;
```

```
// ...2nd GET onwards
rtt = (+new Date - ts);
```

...or simply discard the first measurement.

So, is the instrument measuring web server + client browser performance? Neither! The outcome of these measurements are partly biased by device performance, partly biased by the percentage of the population being sampled, and hopefully mostly dependent on true TCP network latency. In order to fine-tune the instrument, some laboratory statistical data has to be measured regarding the devices performance, most notably the client browser's JavaScript engine performance (as it varies widely between implementations).

Hands on…

Measurement results

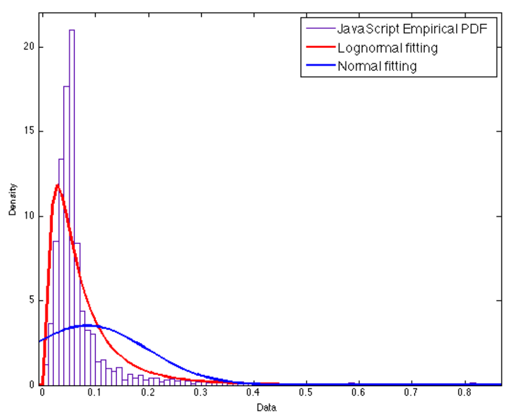

Figure 6.1 - JavaScript ~6K sample normal and lognormal

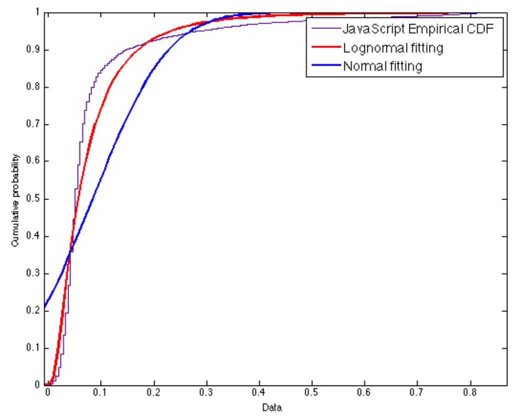

Figure 6.2 - JavaScript ~6K sample normal and lognormal

Figure 6.1 and 6.2 show a ~6K sample data set, from a fixed origin to a fixed destination, collected by the JavaScript tool. The first observation that can be made is that the histogram in Figure 3.1 is not symmetrical: it is a right-tailed histogram, which indicates that the distribution of this sample is not normal. Doing a fitting to match a normal and lognormal distribution, we can visually infer that the lognormal fitting suits better the data set. This curve is coherent with typical lognormal processes, which are those affected by random and independent elements, such as the network factors we suspect introduce uncertainty in this kinds of measurements.

This measurement results encourage us to keep on doing some research about TCP and JavaScript latency relationship. They do not mean that the lognormal distribution is *the best* distribution for this data set. There is still research to be done in searching for the best fitting distribution for JavaScript measurements and analyzing μ and σ, and looking for any relationship between these statistical parameters. We hope to find relationships between distributions and OSes and web browsers, for example (not real values):

HTTP User-Agent Field		**Tester environment**		**Measurement normalization**

Mozilla/5.0 (Macintosh; Intel Mac OS X 10_9_2) AppleWebKit/537.75.14 (KHTML, like Gecko) Version/7.0.3 Safari/537.75.14	**-->**	OS: OSX 10.9.2
		Browser: Safari 7.0.3
			-->	μ shift = -10 ms // shift measurements left 10 ms
				σ = 1.3 // 30% more 'spread' curve

JavaScript and TCP

In order to find out a correlation between JavaScript RTT and TCP RTT, another set of tests was run from the browser's JavaScript engine, and simultaneously sniffed by a TCP sniffer. This way, the same set of measurements were timed from two different scopes.

Some questions we had beforehand:

1. Measurements have shown us JavaScript present lognormal latency results. Will TCP show the same distribution?
2. Will it show similar statistical parameters?

3. Most importantly, can we parametrize any differences between JavaScript and TCP?
4. Which parameters affect mostly these differences? OS? Browser? How the server manages HTTP requests? Network congestion? Time-of-day?

After running the tests and collecting data sets of ~500 samples, the results could be summarized in Figure 4.1 and 4.2[2]:

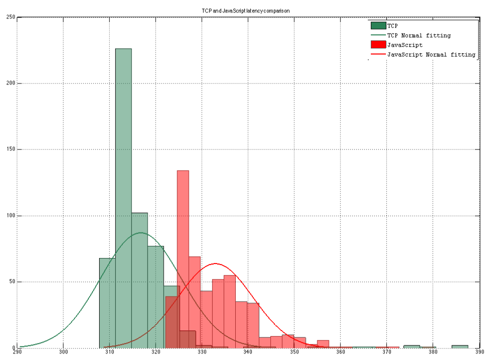

Figure 4.1 - Histogram showing sampled RTTs from the JavaScript tool (red, rightmost) and a TCP sniffer (gree leftmost).

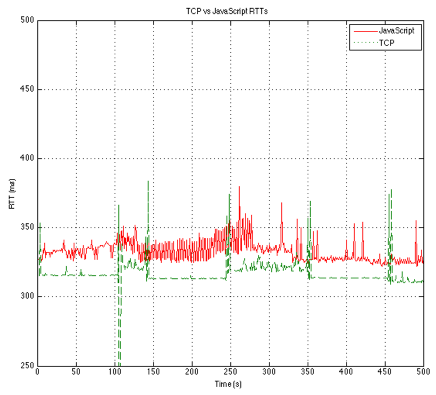

Figure 4.2 - The same data from Figure 4.1 charted vs. time. The red line corresponds to JavaScript, the gree dashed one to TCP.

Figures 4.1 and 4.2 make us believe that JavaScript effectively adds some delay to the browser's TCP connections, as the JavaScript (red) probability histogram is consistently shifted towards higher latency values. As to whether TCP results show a lognormal tendency as JavaScript did, we can only observe that the TCP results are more symmetrical than the JavaScript ones. This leads us to think that JavaScript processing not only shifts the green PDF to the right, but also distorts it in such a manner that the red PDF ends up fitting better a lognormal than a normal distribution.

Fitting a normal distribution[3] to both data sets, the best fit for TCP was μ=316.6 σ=8.6, and for JavaScript μ=332 σ=8.0. The difference between the distributions means was 16.11 ms. It is still to be explored which factors affect this difference, are they clinet-side factor? In which way could these factors be predicted? OS, browser? In future revisions of this document we should explore a bit more about it.

Network congestion

One parameter that affects TCP latency measurements is network congestion. Theoretically, latency measurements are closely tied to the network congestion at the moment of observation, the more congestion, the greater the latency values.

Having this fact in mind, we decided to inspect the latency measurements gathered and see if there is a direct correlation between these two attributes. If this relationship is found, we can predict the delay introduced in the measurements due to congestion and mitigate errors in measurement. Any direct relationship should show greater latency values at peak traffic hours. However, our latency measurements show that there is no

[2] Tests were ran on Safari 7.0.3 browser on OS X 10.9.2, on a wired ethernet connection, and sniffed by the *tcpdump* utility.

[3] Lognormal distributions are 0-based, and important metrics such as μ difference can't be correctly estimated. For a preliminary analysis, we chose to fit data to a normal distribution.

such relationship. The relationship between RTT and time-of-day can be seen in Figure 5.1, 5.2, and 5.3.

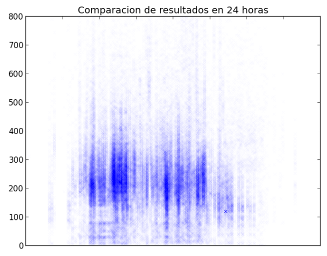

Figure 5.1 - Regional RTTs

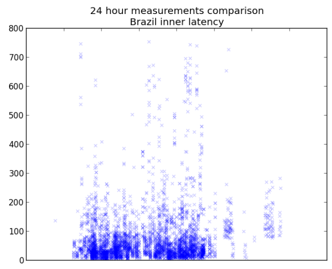

Figure 5.2 - Brazil inner RTTs

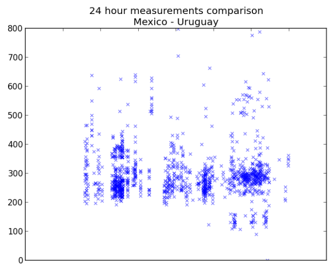

Figure 5.3 - Mexico <--> Uruguay RTTs

As Figure 5.1, 5.2, and 5.3 show, there is no direct relationship between time-of-day and latency measured, as measurements look like 'white noise'. These three different scenarios try to represent the LAC Internet in three different ways, having the whole region considered (5.1), only a specific country (5.2), and a pair of countries (5.3).

Considering that LAC Internet traffic is cyclical in a 24-hour basis [2][4], we could assume there is a direct relationship between time-of-day and network congestion. Having this in mind and considering the 5.X charts, we tend to think there is no direct relationship between LAC Internet congestion and out latency measurements.

Browser tests

As mentioned above, JavaScript execution is tied with the browser's implementation of the JavaScript engine. Because of this, different browsers result in different RTTs, so the question is, is there a way to determine a "normalized" JavaScript latency? Lets say, could we assume that, for example, Google Chrome browser results have a determined X milliseconds shift of its PDF? Is there some relationship in the result's variance?

Running some quick (~1000 samples) tests from three different browsers (on the same OS, at the same time), we found that browsers show some difference in their curve's μ and σ. Test results are shown in Figure 7:

[4] For IPv6, network traffic at IXPs is not as cyclical as for IPv4 [3], but considering most of the measurements are still done over IPv4, we could discard the acyclical behaviours at the moment.

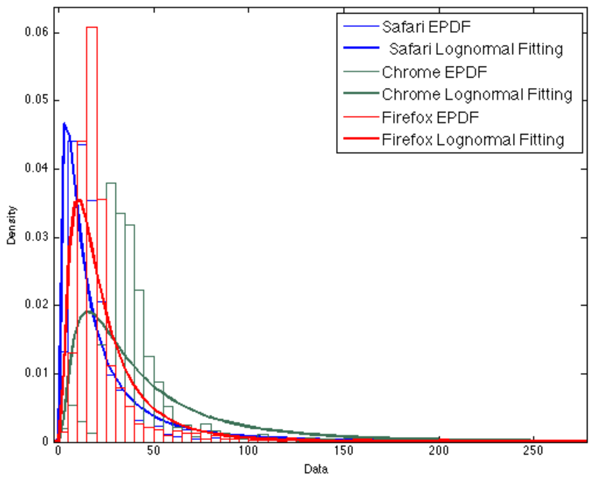

Figure 7 - Differences between browser's perceived latency.

Figure 7 shows clear differences between browsers behaviour for this data set. All histograms show a lognormal distribution, so the parameter to be looking at is the *median* values and how they differ from browser to browser.

At the moment no statistically relevant conclusions can be reached, but after doing some more research and having enough amounts of data we hope we'll be able to draw more interesting conclusion about the browser's JavaScript engine behaviour and their differences.

Conclusions

Field of application

Latency measurements have a strong relationship with network connectivity: the lower the latency values, the better the connectivity. Usually longer paths require more time for the same information to travel, sub-optimal routing makes this path differences manifest into time differences. Network connectivity is an attribute that not only depends on network topology, but also depends on agreements between network operators and other agents (other Internet Service Providers, Internet Exchange Points, backbone providers). Therefore, poorly connected ISPs will show greater latency values.

Given this scope, by making "mesh" latency measurements, that is from different origins to different destinations, a connectivity matrix between AS, countries, or regions could be built. Also, path difference could be detected as a consistent difference in latency values. Let's check the following example, with samples taken with the Javascript tester between Chile and Uruguay (~1300 km apart):

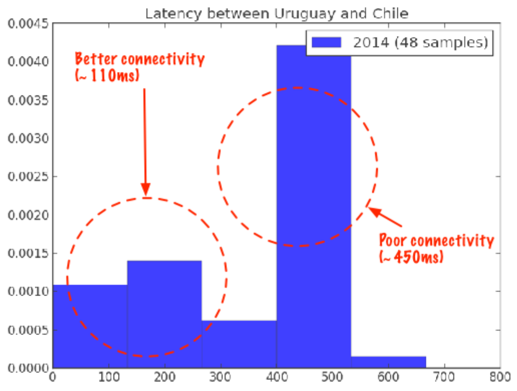

Figure 6.1 - Path difference in latency measurements between Chile and Uruguay in 2014

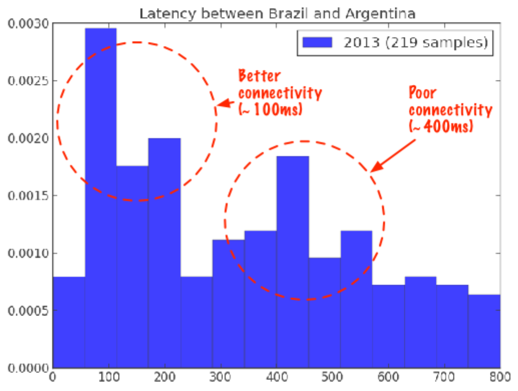

Figure 6.2 - Path difference in latency measurements between Brazil and Argentina in 2013

The histograms in Figure 6.1 and 6.2 shows two clear peaks, one around 100 ms and another around 400 ms. This kinds of results lead us to think about information flowing between two countries via two very different paths: the slower taking 4x the time to travel than the fastest one. The countries in this example should consider making their connectivity better!

Conclusions

In this document we have covered some reasons for which JavaScript might help network measurements and analysis. Javascript has always been considered a web front-end design programming language, but tests have shown that when pushing its capabilities to the edge it can help harvesting of information from very diverse sources. Pushing the language and its context to the edge implies making strong assumption about many elements, from which this paper focused on browser performance.

Despite being in an early stage, the JavaScript tester has demonstrated that by gathering results in a "mesh" fashion it can help to get to know network latency values in a realistic way, specially when the networks scanned are big and somewhat divided (different network operators, bad routing policies).

Nevertheless, this new approach to measuring latency doesn't substitute traditional latency measurement techniques. The JavaScript tool provides information on how the end-user perceives network latency, as an OSI layer 4 (TCP) attribute. Using the JavaScript tool is a way to complement those traditional results, originating from other ICMP , TCP, and UDP tools.

We strongly think that the kind of information gathered by this type of tool is very valuable for encouraging more regional connectivity agreements, infrastructure investments, and local content. A widespread use of tools of this kind will eventually lead to gather information of good quality and develop a better regional Internet.

References

1. Touch, J., Heidemann, J., and Obraczka, K., Analysis of HTTP Performance,

USC/Information Sciences Institute.

2. Pontos de Troca de Trafego do Brasil (PTTMetro) - Comitê Gestor da Internet no Brasil (CGI.BR). IPv4 Weekly Graph (30 minute average) http://sp.ptt.br/images/pix/agregado_bps-weekly.png

3. PTTMetro - CGI.BR. IPv6 Weekly Graph (30 minute average) http://sp.ptt.br/images/pix/agregado-ipv6_bps-weekly.png