WPI-CS-TR-10-14 April 2012

JavaScript and Flash Overhead in the Web Browser Sandbox

by

Murad Kaplan
Mihajlo Zeljkovic
Mark Claypool
Craig Wills

Computer Science Technical Report Series

WORCESTER POLYTECHNIC INSTITUTE

Computer Science Department
100 Institute Road, Worcester, Massachusetts 01609-2280

Acknowledgements

This material is based upon work supported by the National Science Foundation under Grant No. MRI-0959441. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the National Science Foundation.

Contents

1. Introduction

We use the Web browser sandbox methodology from [1] to estimate the round-trip, download time and upload time using JavaScript and Flash execution from within a browser. In this report, we measure the browser sandbox overhead of using these methods to establish a baseline for later work on the project to estimate application performance.

There are four measurement techniques we use, two explicit and two implicit. JavaScript XMLHTTPRequest and Flash URL Request are explicit techniques for retrieving files from the origin server. JavaScript DOM and Flash Load Policy File are implicit techniques used for retrieving files from any third-party server. Brief descriptions of these techniques are given in the following with more details on each available in [1].

1.1 JavaScript XMLHTTP request - XHR

The explicit technique for JavaScript is to retrieve the given URL for an object and record the duration of the download process using the XMLHTTPRequest. This technique can be used to GET or PUT an object, but can only be used in conjunction with the origin server.

1.2 JavaScript DOM Object - DOM

The implicit JavaScript technique we use is to load an object from a server into a <form> tag within the Document Object Model of the current page. We record the start time when we change the source of the form and record the end time when loading is done, signified by the execution of the JavaScript "onload" event. This technique is implicit because it indirectly measures the download time of the object. It is of particular interest to our project because it can be used in conjunction with known objects from any server on the Internet, not just from the origin server.

1.3 Flash URL Request - URL

Flash code first needs to be compiled into a Flash SWF file and then inserted into an HTML page through an <object> or <embed> tag. The explicit technique for downloading an object with Flash is to record a start time and then execute a function to download a file. When the function is done, an event handler function is executed and the end time is recorded. This technique can only be used with objects obtained from the origin server.

1.4 Flash Load Policy File - LPF

Flash is restricted by the same-origin policy by default and cannot retrieve files from other domains. This limitation can be changed by placing a policy file named acrossdomain.xml in the server root directory and listing all domains that provided access. When trying to retrieve a file, Flash first gets policy file and then requests an object. If the domain where the Flash is hosted is on the policy file list then the file will be retrieved. If not, an exception will be thrown. We use this mechanism to measure the download time for an object from an arbitrary server by trying to retrieve a file from the server as a policy file and time how long it takes for an exception to occur.

In network communication, a client and server exchange packets during a session. A packet goes through several phases from the application layer down to the physical layer and vice versa. The scripting languages stand above the application layer, and the time between the packet arriving to the physical layer and the time reported by web browser is the overhead we want to analyze. We found out that this time depends on many factors including web browser type and version, operating system, method (JavaScript or Flash), and hardware (CPU, RAM).

In this report, we compare JavaScript and Flash performance for measuring download, upload and round-trip times of different servers and try to understand the overhead of our methods. We want to determine the difference of the actual time and the one that is obtained by a Web browser. To calculate the overhead, we use JavaScript and Flash for high-level data measurement and compare to low-level data obtained by monitoring network packets transfer using a native executable performing similar upload and download operations. This code was written in C and we use its output as the "ground truth" in our experiments.

2. Experiment Design

We set up environments to control bandwidth and delay and compare the predicted performance times to actual times over a range of controlled network settings. As showed in Figure 1, two machines were used that run the following Web browsers: Chrome v12, Internet Explorer v8 and Firefox v4. The machines sent network traffic through a Linux PC, configured with `netem` to control bandwidth and delay to provide a network environment of a typical residential ADSL link: 1 Mb/s download and 256 Kb/s upload with 50msec delay. Table 1 shows hardware and software specifications for each machine.

Table 1. Machines Specifications

	Computer 1 (Desktop)	Computer 2 (Laptop)
Hardware	Dell Desktop CPU: Intel® core i7 CPU 870 @ 2.93GHz RAM: 8.00 GB Network Card: Intel 82578DM Gigabit Net Connection	Dell D620 CPU: Intel® core 2 Duo CPU T5600 @ 1.83GHz RAM: 2.00 GB Network Card: Brodcome netXtream 57xx Gigabit Controller
Software	OS: Windows 7 64 bit, Ubuntu	OS: Windows XP 32 bit, Ubuntu

3. Methodology

For each machine, using the two operating systems and Web browsers installed on them, we ran three different tests -- download, upload, and round-trip time tests. For each configuration, we ran the four methods for JavaScript and Flash except for the upload test. Only the XHR and URL techniques can be used to upload objects to a server.

We compared our result using these methods to a baseline measurement based on a program written in C we wrote to stream packets between the two machines and server. The code works on the transport layer, designed to minimize any overhead that may be caused by the application layer.

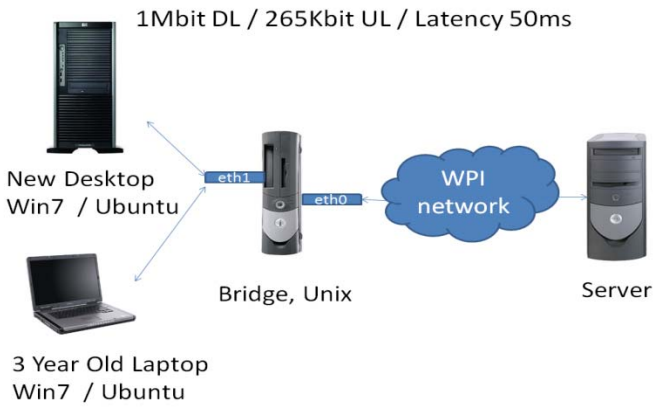


Figure 1. Experiment Setup

3.1 Download Test:

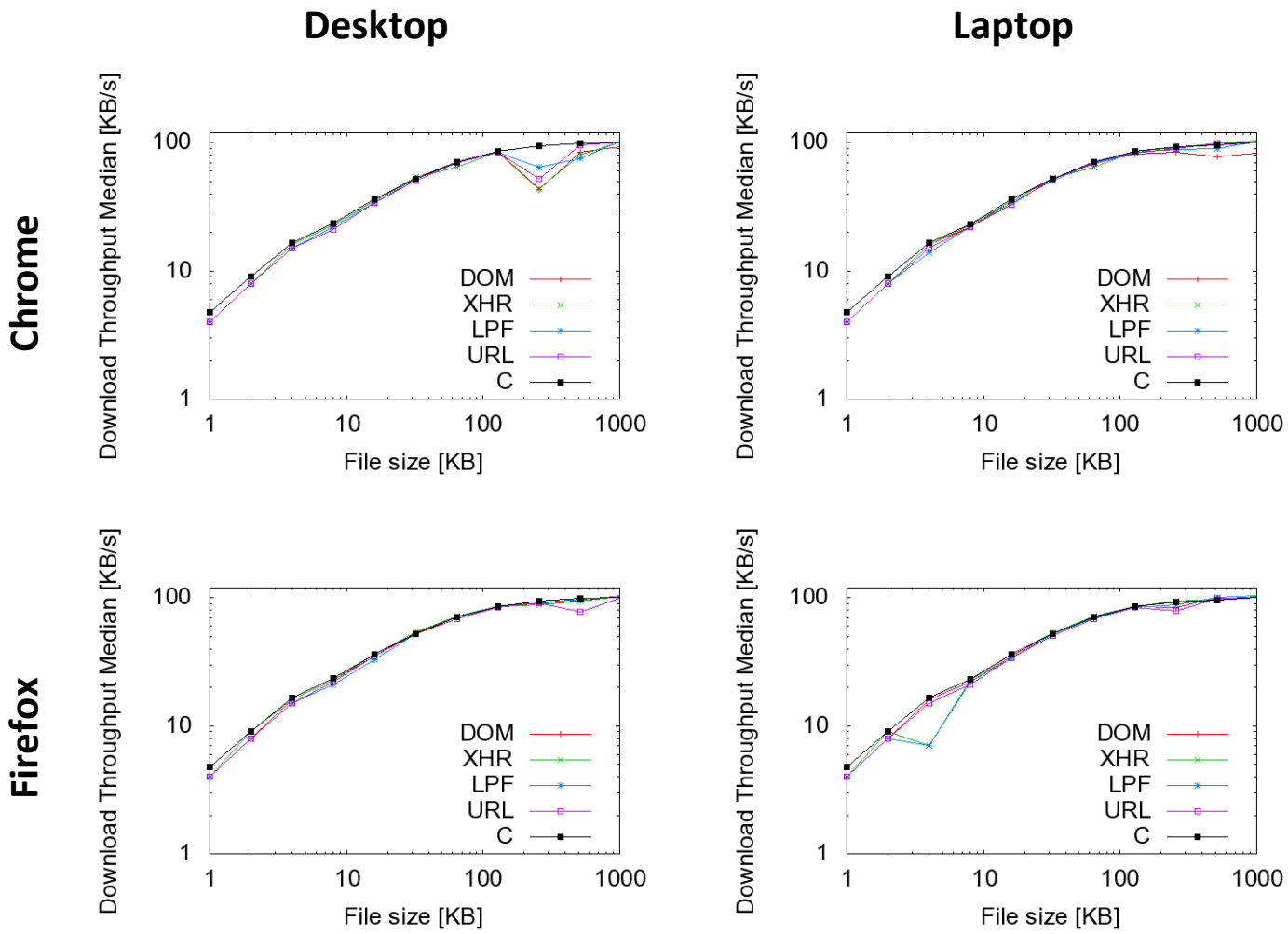
We chose a range of object sizes from 1 KB to 1MB. We downloaded objects of each size once using each method and then downloaded the next size larger object and so on. After downloading the 1MB object we restarted the browser and repeated the test again. This approach caused the cache to be cleaned each time to make sure that the browsers are retrieving the objects from the server and not from the cache. We repeated this approach 5 times in each machine, each operating system, and each browser and calculated the average, median, and standard deviation for each object size.

3.2 Upload Test:

Same as the download test, we chose a range of object sizes from 1 KB to 1MB. We uploaded each size to the server once using each method and then uploaded the next size larger object and so on. We repeated this approach 5 times for each machine, each operating system, and each browser and calculated the average, median, and standard deviation for each object size.

3.3 Jitter Test:

In this test, we download a 1KB object to make sure that it would be carried in one IP packet, 50 times for each browser. We calculated the cumulative distribution of each run.



Figures 2-5. Median Download for Ubuntu OS

4. Results

4.1 Download Results:

Figures 2 – 5 show the median of the download throughput for the Ubuntu operating system on the two machines using the four methods and C code. For consistency, all results are shown in terms of median values. The mean values for all measurements are generally within a few percent of the corresponding median value with the largest discrepancy between mean and median about 6% of the reported median. Figures 6 to 11 show the median download throughput for the Windows 7 operating system on the two machines. The Y axis is the median throughput of the five runs for each object size and the X axis represents the 11 object sizes. As can be seen, in Windows 7 the JavaScript methods outperform (incur less overhead) than the Flash methods especially in with the small files. We can see better performance for all methods in Chrome, but a larger difference

between methods in Firefox and Internet Explorer. In Ubuntu we generally see more consistent performance except on files larger than 100KB, where some methods exhibit reduced performance.

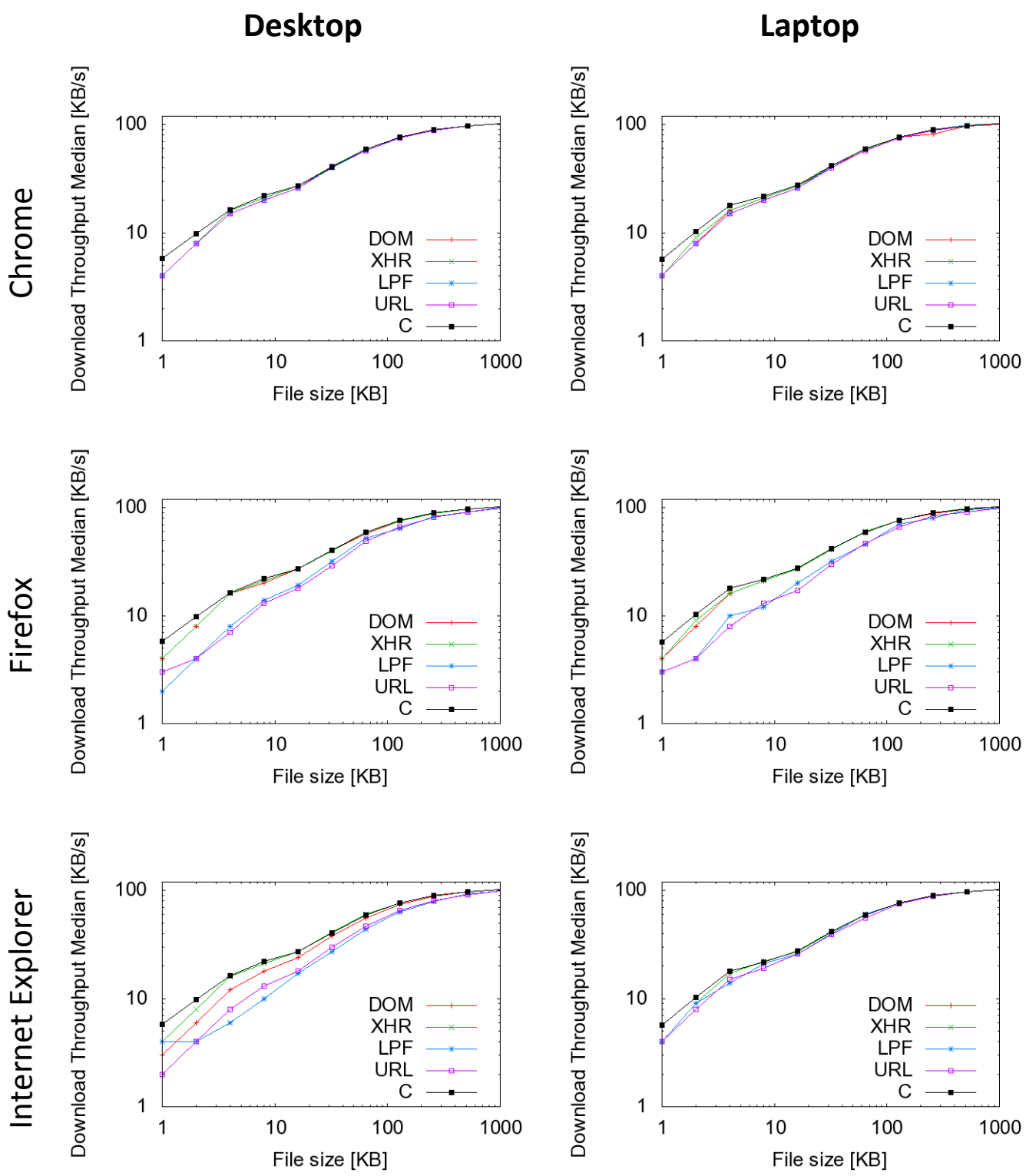
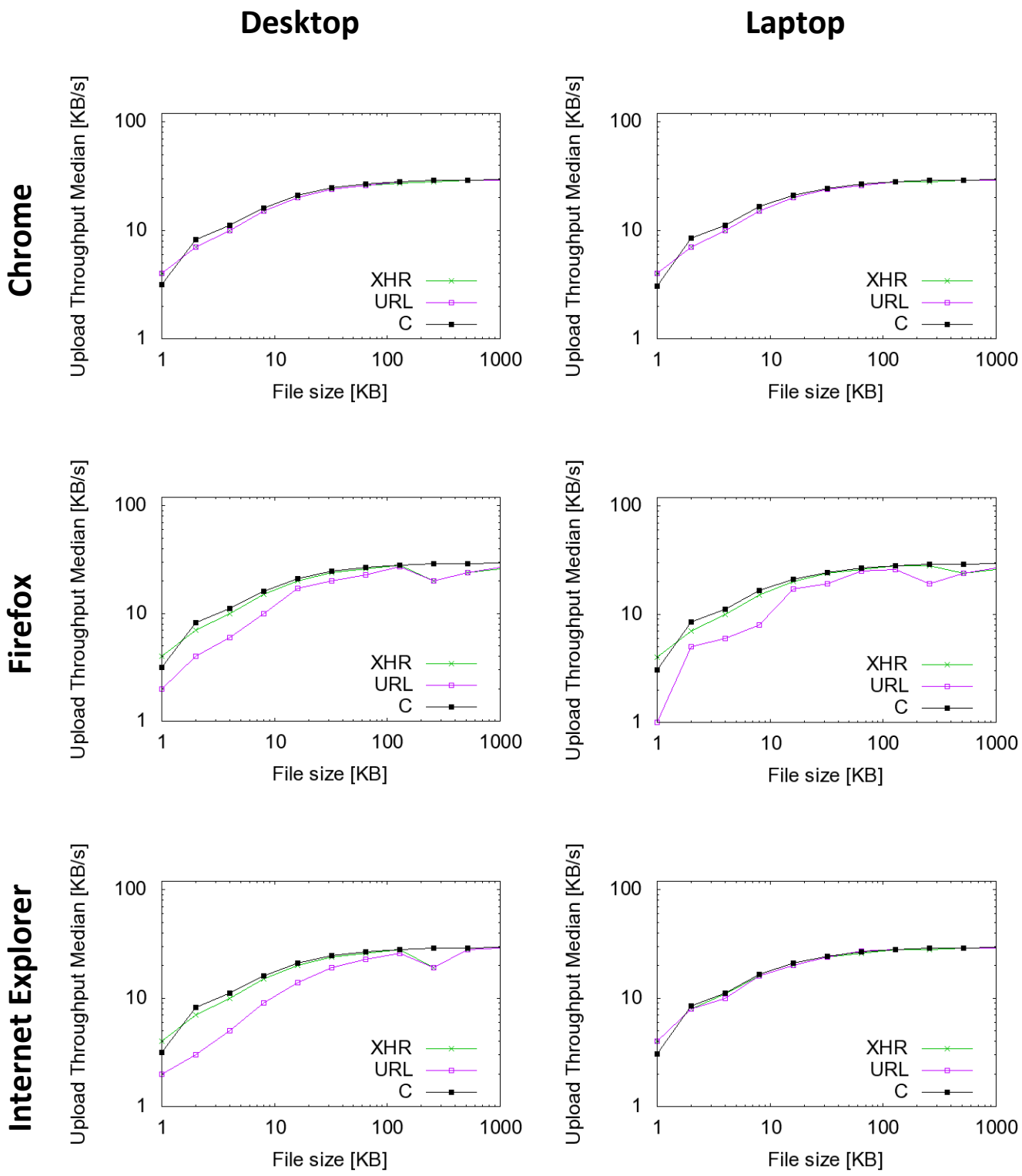
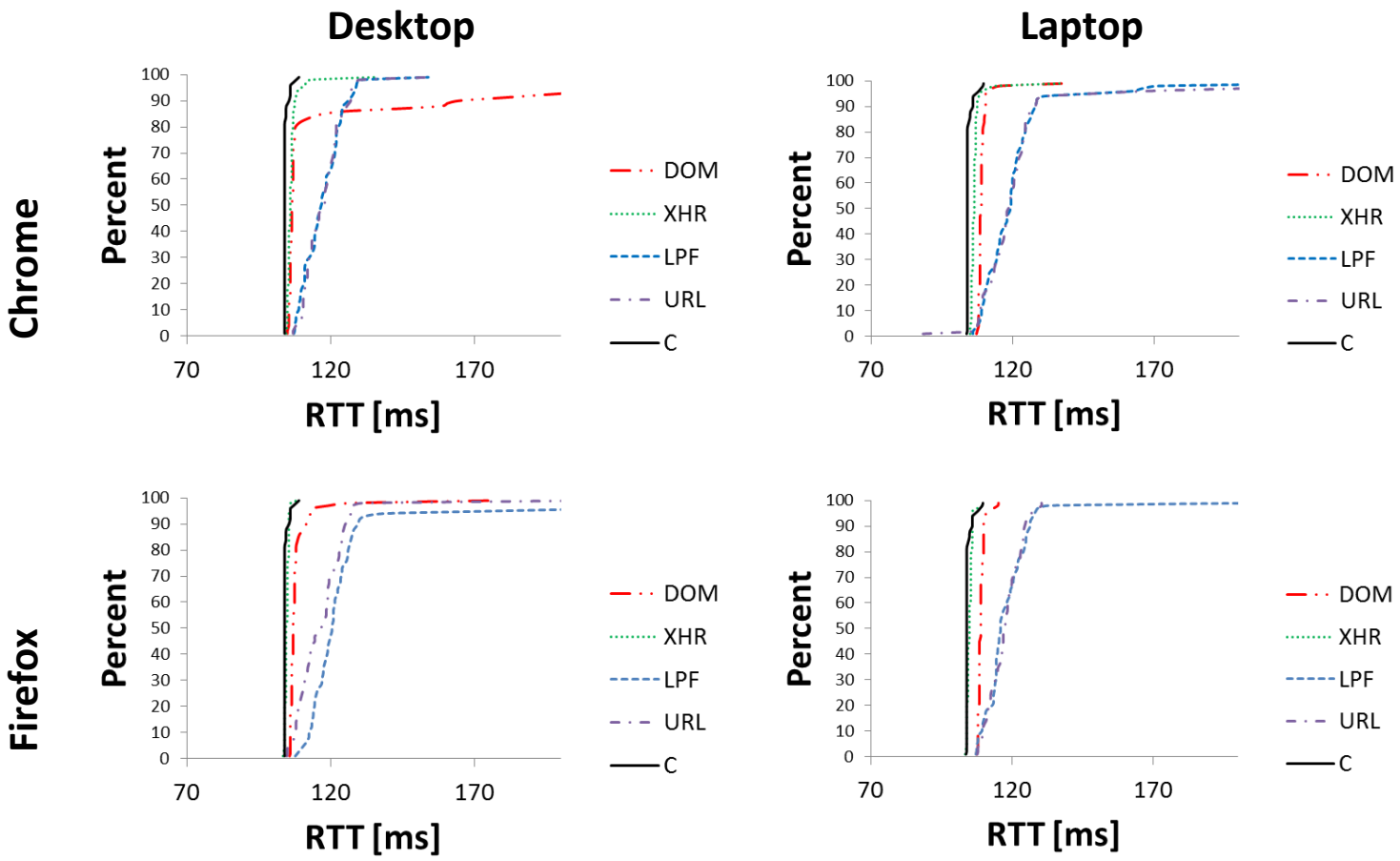


Figure 6 – 11. Median Download for Windows 7 OS

.



Figures 12 – 17. Median Upload for Windows OS



Figures 18 - 21 Jitter CDF for Ubuntu OS

4.2 Upload Results:

As previously mentioned, we only have two methods that can be used to upload objects to our server. These methods are XHR and URL. Figures 11 – 17 show the median of the upload throughput for Windows operating system on the two machines using the two methods and C code. The Y axis is the Median upload throughput of the five runs for each object size and the X axis represents the 11 object sizes. We did not include results from the Ubuntu machines as we experienced difficulties using our C code for uploading packets to the Ubuntu client machine. Again, we see better performance for Chrome compared to the other browsers. We also see that JavaScript method shows a better performance than the Flash method.

4.3 Jitter Results:

Figures 18 – 21 shows the cumulative distribution function (CDF) for download of a small file for the Ubuntu operating system on the two machines using the four methods and C code. Figures 22 – 27 show the CDF for download of a small file for the Windows 7 operating system on the two machines. The Y axis is the cumulative distribution of the 50 round-trip time measurements to the server and the X axis represents the round trip time in milliseconds.

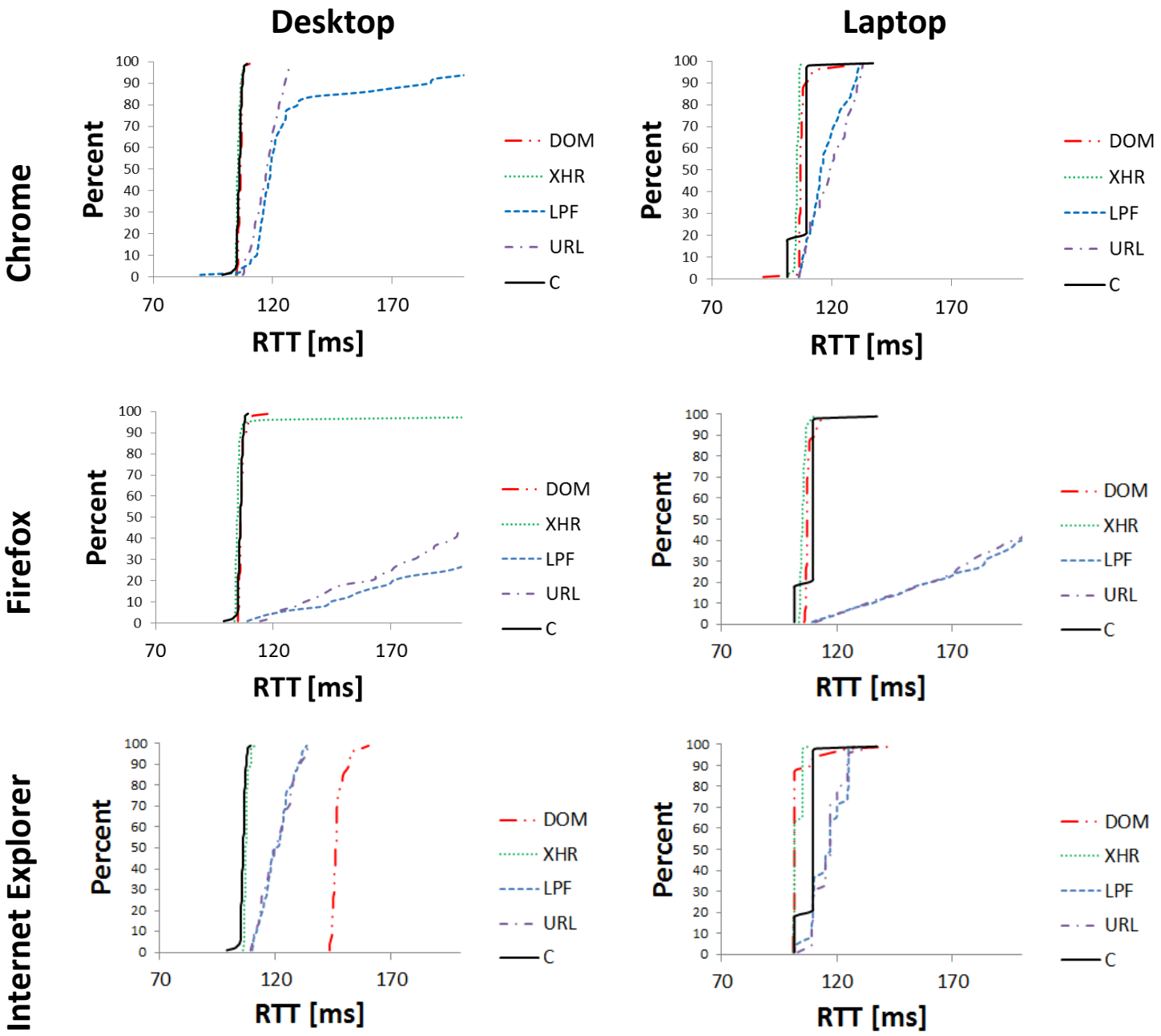


Figure 22 – 27. Jitter CDF for Windows OS

In general, the JavaScript methods are the best for being closest to the ground truth (C code), although Internet Explorer exhibits a constant overhead when compared to the baseline.

5. Conclusion and Future Work

In this work we systematically measure the overhead from Web browser sandbox methods introduced in previous work to measure network performance. As can be seen from the results on two different machines using two operating systems and three popular Web browsers, these sandbox methods largely provide accurate measurements of the network performance.

Our results showed that JavaScript methods generally give better results than Flash methods. We saw that JavaScript XHR is closest to the ground truth for measuring download, upload and round-trip time. We also saw that JavaScript DOM produces constant results for round-trip time and can therefore be used for measuring jitter. Flash methods can be close to the ground truth for download and upload, but they do not outperform JavaScript XHR when comparing against ground truth.

We plan to these results as a basis for providing estimates of application-level to users of our How's My Network Web site. The methods that can be used with arbitrary servers are particularly attractive as they allow us to perform measurements directly to servers of interest to users rather than being restricted to just the origin server.

6. References

[1] Artur Janc, Craig E. Wills, and Mark Claypool. Network Performance Evaluation within the Web Browser Sandbox, http://www.wpi.edu/Pubs/ETD/Available/etd-011909-150148/unrestricted/artur-janc-msc-thesis.pdf, January 2009

[2] Firebug, http://getfirebug.com/

[3] Wireshark · Go deep, http://www.Wireshark.org/