NETWORK PERFORMANCE EVALUATION IN A WEB BROWSER

Artur Janc, Craig Wills and Mark Claypool
Department of Computer Science
Worcester Polytechnic Institute
100 Institute Road
Worcester, MA 01609, USA
email: {artur,cew,claypool}@cs.wpi.edu

ABSTRACT

The Web browser is increasingly used for emerging applications such as online games and streaming audio/video. End users seeking better quality for these applications need methods to ascertain network performance in a simple manner. This work investigates methods for measuring network performance through a Web browser and Flash plug-in while requiring only the minimal user participation of navigating to a Web page. A focus of the work is to complement explicit measurement techniques with new implicit measurement techniques we develop that allow performance measurements to arbitrary servers on the Internet and allow the measurement of delay jitter.

Results from the new techniques are compared against existing techniques for measuring download, upload, round-trip times and delay jitter across different operating system and browser environments. The results show that JavaScript and Flash techniques can be used to reliably estimate client download throughput and RTTs to arbitrary hosts, which could lead to customized tests and reports specifically for multimedia, game and other application sites of interest to users.

KEY WORDS
network measurement, Web browsers.

1 Introduction

The increase in network capacity to the home has prompted the growth of emerging applications such as streaming audio and video and network games. While traditional applications are relatively forgiving of modest latencies and a wide range of bandwidths, multimedia applications, particularly streaming videos, perform best with high bandwidth, and interactive multimedia applications, such as network games and VoIP, require low amounts of latency and delay jitter.

However, as the use of home and public networks has grown the performance provided to users of traditional and multimedia applications is not well understood. Network researchers have means to measure performance from well-known platforms such as PlanetLab [7], but this platform of nodes are largely located at university and research labs and not representative of the networking vantage points seen by most users on the Internet. Platforms such as NETI@-home [11], DIMES [10] and DipZoom [14] allow measurements from any node in the Internet, but the platforms provide little incentive for the general populace to participate.

As an alternate, we have a current project to develop a *user-centered* network measurement platform, called *How's My Network* (*HMN*), that provides incentives via games and feedback so the public perceives benefits in participating. The goal is to develop accessible measurement tools for users in a form that will not only provide meaningful performance feedback on applications of interest to users, but also help researchers understand performance from the vantage points of most users on the Internet.

Providing appropriate incentives while limiting impediments for use is a challenge. Our assumption is that any approach requiring the installation and use of a software tool is too big of an impediment for most users. Therefore we have chosen to focus on the Web browser as a platform for performance measurement. Web browsers have become a standard home user application, supporting traditional text-based use, like email and file transfer, while increasingly being used to access streaming media, such as Internet radio and social networking videos. Flexible plugin environments capable of supporting multimedia are commonplace in nearly all Web browsers. Due to their ubiquitous and flexible nature, Web browsers make a natural candidate for an end-user network performance platform.

The challenge for this platform choice is whether the browser provides enough capabilites to perform a range of performance measurements. In related work, we are investigating the potential of a signed Java applet for home network measurement [8]. Signed Java applets provide many capabilities, but require users to accept a certificate in order to be used.

In this work, we examine the more constrained environments of JavaScript and Flash for determining end-user network performance characteristics. The advantage of these environments is they are available in most browsers and require no user intervention. Unfortunately, the "sandbox" environment of JavaScript and Flash present several challenges to network measurements. Specifically, they have: 1) a lack of direct access to the network layer, mean-

ing only HTTP (and thus only TCP, no UDP or ICMP) traffic, and browser scripts receive no information about packet timings; 2) scripts that can only connect directly to the original server, making it difficult to gather performance information from the client to a wide-range of Internet servers; and 3) no reliable persistent storage on the client is allowed, meaning performance data must be transmitted, analyzed and displayed immediately.

Despite these challenges, this paper presents the design of a Web browser-based testing platform, able to ascertain network performance measures relevant to a range of applications and present results to the client while storing the results for long-term analysis by network researchers. Network measurement techniques are implemented to actively measure the network conditions of the connecting client. The presented work has two main goals.

First, to assess the potential for a Web browser to measure network characteristics of interest to traditional and multimedia applications. Commercial "speedtest" services [3, 6], which typically use Flash, are available but report only download and upload throughput and the round-trip time (RTT) from a client to a speedtest server. However these services do not provide measurements to the Web, streaming and game servers actually being accessed by users nor do they provide measurements on network jitter. In addition, aggregate data about a users' network performance is limited. In contrast, we have developed solutions that overcome these limitations.

Second, the work seeks to compare and validate performance measurements across a range of operating system and browser environments. Users access the Internet via a variety of environments and it is important to understand if there is consistency in measurements amongst these environments. In achieving this goal, the accuracy of Web browser-based measurement techniques are not only compared with each other, but with standard measurement tools.

"Our work makes a number of contributions in better understanding home network performance for traditional and emerging Internet applications such as network games and streaming media. Specifically we have:

1. designed, implemented and evaluated several scripting techniques to estimate upload/download throughput and RTTs in JavaScript and Flash;

2. compared the relative accuracy of these techniques against each other across different operating system and browser environments;

3. developed implicit measurement techniques that allow reliable measurements of download throughput and RTT to arbitrary third-party servers;

4. successfully tested a technique to measure the jitter of a "streaming" Web server with clients in JavaScript and Flash; and

5. performed a distributed set of browser-based network measurement tests along with analysis of the results.

The remainder of this paper is organized as follows: Section 2 outlines the approaches developed for measurement; Section 3 describes the platform for testing these approaches; Section 4 presents the results from controlled and open user testing; and Section 5 concludes with a summary of the work along with directions for the future.

2 Approach

This study focuses on the capabilities provided by the JavaScript and Flash environments to evaluate end-user network performance characteristics. JavaScript and Flash have become ubiquitous in the recent years due to a large number of Web 2.0 applications based on both technologies, and have reached adoption rates of over 95% among Internet users [2, 12]. JavaScript and Flash are also available for almost every browser (Internet Explorer, Firefox, Safari, Chrome) on every major operating system (Windows, Linux, Mac), and are supported by many mobile devices, including PDAs and mobile phones.

Investigation of both the JavaScript and Flash network interfaces to transfer data to and from Internet hosts enables an estimate of the client's download and upload throughput, and send/receive timing provide RTT. Based on the specifics of the invocation of network communication, available methods are divided into two categories: *explicit* where the browser provides an object with methods to retrieve data from a network URL using HTTP; and *implicit* where the browser dynamically includes or tests the availability of resources (such as images or scripts) causing subsequent retrieval of data, possibly from a third-party server.

Table 1 summarizes the performance measurement capabilities of the four techniques we developed using explicit and implicit approaches with JavaScript and Flash. Explicit techniques to measure upload/download throughput and RTT are present in existing speedtest services. However as shown in the table, we have extended the explicit techniques to allow the measurement of delay jitter with the origin server. In addition, we have developed implicit techniques for both JavaScript and Flash that allow us to measure download throughput and RTT measures to be obtained for both the origin and third-party servers. Details of these techniques are outlined below with additional detail available in [5]. The code for these methods has been packaged and is available from a public source code repository [4].

2.1 Explicit Communication Methods

Recent versions of JavaScript and Flash provide objects that facilitate asynchronous retrieval of content from the

Table 1. Network Performance Measurement Capabilities by Technique

Technique	Technique Category	Origin Server				Third-party Server		
		Dwnld Tput	Upload Tput	RTT	Jitter	Dwnld Tput	Upload Tput	RTT
JavaScript: XHR	Explicit	✓	✓	✓	✓			
JavaScript: DOM	Implicit	✓		✓		✓		✓
Flash: URLRequest	Explicit	✓	✓	✓	✓			
Flash: loadPolicyFile	Implicit	✓		✓		✓		✓

origin server.[1] Such objects are commonly used to request data from the server "in the background", in response to or in anticipation of a user's action. In JavaScript this capability is provided by the XMLHTTPRequest object, also called *XHR*, mandated by a W3C standard [13] and available in all major browsers. In Flash, the same functionality is provided via multiple interfaces in the `flash.net` package, including the `URLRequest` and `URLLoader` classes [1]. We refer to this family of functions and objects as *URLRequest*.

Both JavaScript and Flash go through four phases: 1) instantiating the request object; 2) setting appropriate parameters—the only required parameter that must be set is the URL, and typically the type of the HTTP request, such as GET or POST; 3) setting function handlers to receive events about the request status—in JavaScript, the request object's `onreadystatechange` handler receives updates at each phase of the request as it is processed, while in Flash, the `addEventListener` function of a URLLoader object allows the setting of function handlers for various event types; and 4) sending the request.

By recording two timestamps, just before sending the request and just after the response has been fully received, the communication duration can be measured and used to determine throughput and RTT to the origin server. The RTT is obtained by first requesting a small object to establish a persistent connection and then retrieving the small object again, which is done in a single round trip.

An important limitation of XHR and URLRequest is that they can only send requests back to the origin server as defined by the same-origin policy [9]. While useful network information can be gathered with this constraint, it limits the generality of the explicit communication techniques.

2.2 Implicit Communication Methods

If measurement techniques can be developed that are not constrained by the same-origin policy then performance measurements could be customized to the specific interests of users. Using this observation as motivation we successfully developed two implicit measurement techniques.

The JavaScript Document Object Model (DOM) allows scripts to dynamically insert resources into the current document. In addition, the DOM can receive events related to user actions (such as `onclick`, `onmouseover`) and page/resource load status (such as `onload`, `onunload`, `onerror`), where the programmer can define JavaScript handlers for such events. Combining the functionalities allows measurement of the time from request to response through three steps: 1) record the start time; 2) dynamically insert a resource in the page with a custom `onload` handler, invoked when the resource has finished downloading; and 3) record the end time in the `onload` handler and calculate the duration of the download process.

Flash provides a similar method to request external resources by treating them as policy files, requested through the `loadPolicyFile()` function (LPF). While Flash expects URLs used with this function to be policy files, the function can retrieve any known object. Using this functionality we: 1) record the start time; 2) invoke `loadPolicyFile()` (done in background) on a known URL; 3) retrieve URL with `URLRequest`; and 4) record the end time when `URLRequest` fails due to security failure because `loadPolicyFile()` completes and does not grant the needed access.

These implicit communication mechanisms are not as obvious to implement and are more susceptible to programming errors than well-described applications of the explicit methods. Moreover, these methods do not provide any information about the current state of the request, and do not signal any intermediate events. However, a significant advantage of implicit techniques over explicit techniques is their ability to retrieve resources from external domains, making it possible to determine the download throughput and RTT between the client and any Web server on the Internet.

2.3 Measuring Packet Jitter

In addition to analyzing the browser's potential to measure throughput and RTTs, we also evaluated its capability for gathering delay jitter for scenarios where data is streamed

[1] The origin server is the host from which the current page (HTML document) was received.

across the network.

While browsers have no mechanism for sending or receiving UDP traffic to emulate a streaming UDP protocol, it is possible for an HTTP server to send data at regular intervals to simulate streaming using TCP at an application-controlled rate. Combined with the ability of explicit communication mechanisms to receive updates about the status of a request as it is being processed, it is possible for the browser to receive information when a new chunk of data has arrived.

Using intermediate event features in JavaScript and Flash, packet reception variability in a simulated streaming protocol can be measured by recording times at which data is received by the browser. Such information can be useful for predicting performance of video and VoIP applications, which are increasingly being streamed over Web platforms.

For the server-side component of the jitter test we implemented a basic Web server in Python. The server sends data in a "streaming" manner by breaking data into fixed-size chunks and sending at periodic intervals. The TCP `NO_DELAY` option is used to ensure that packets are sent immediately after data is written to the network socket.

The client script, which requests a resource from the streaming server, must be able to receive events about incoming data as it is being transferred. Therefore, only the explicit download techniques—XMLHTTPRequest and URLRequest—have the potential to provide the necessary information; due to browser incompatibilities in the implementation of events in the JavaScript XMLHTTPRequest object,[2] our jitter client is implemented in Flash.

Flash allows programmers to set handlers for events related to the status of the `URLRequest` object, including `ProgressEvent.PROGRESS`, which is triggered when new data is available during a download process. By handling progress events Flash can determine the times when data is received from the server. By comparing this information to the server's log of timestamps when data was sent the variability of data availability can be determined, and hence packet arrival times, assuming receive buffering is minimal.

3 Measurement Platform

As shown in Figure 1, our measurement test system is implemented as a Web-based application available on a server on the WPI campus network. A client connecting to the test Web site is served the script files (in Flash and JavaScript), and a static HTML page for displaying results. Implicit techniques may cause content to be retrieved from third-party servers.

For throughput and RTT measurements, tests are set to repeat every five minutes to allow for several tests runs over time if the user so chooses. A single test consists of a measurement of the download and upload throughput (using 2MB of data in each case) and the RTT (using a 1 byte file) for each of the available techniques (XHR, URLRequest, DOM and LPF), to either the server or a third-party host. After each individual measurement, the client sends a summary log of the result to the server and performs the next test. As throughput and RTT results are gathered, the HTML page provides the user with information about each test result. A long-term goal of our work is to provide this information in a manner that shows users how it will impact the performance of their applications of interest.

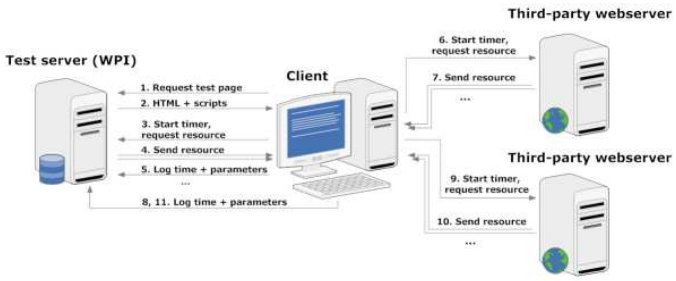

Figure 1. Test Platform System Design

Table 2. Operating System and Browsers Used in Controlled Study

OS	Browser and Version	Flash Version
Windows XP	Internet Explorer 7.0	9.0
Windows XP	Mozilla Firefox 2.0	9.0
Windows XP	Google Chrome 0.9	9.0
Ubuntu Linux 8.04	Mozilla Firefox 3.0	9.0

For jitter tests, the client initiates the request for a streaming resource and keeps track of intermediate events related to the transfer. After the download process is complete, the client sends a list of timestamps corresponding to these events to the server. The server combines its own log of packet sending times and the user-submitted data to display statistics and graphs of packet reception variability to the user.

4 Results

With the measurement platform in place, two sets of tests are performed: 1) a controlled test using many platforms from a single residential location; and 2) an open test where users were invited to execute the tests from any platform and any Internet location.

4.1 Controlled Tests

To perform initial validation in a controlled environment, we executed a test in several browsers using all of the de-

[2] Internet Explorer does not support intermediate event notifications for XMLHttpRequest objects. This feature is supported in Firefox.

veloped measurement techniques as well as standard non-browser measurement methods as a measure of the "true" value during the period of measurement.

The client hosts were located on the US West Coast in a campus residential network connecting to the server on the WPI academic network. The clients used four browser/OS combinations as shown in Table 2.

Tests were repeated every 12 minutes for a three-hour period starting at midnight PST. Care was taken to ensure that no two tests were executed at the same moment by spacing out the start time of the measurements in each browser.

Download throughput results were obtained for each of the four developed techniques on each of the four browser & OS platforms. Validation was done by using the `wget` tool from the Linux client to download the same file from the Web server. The mean results for each method and their 95% confidence intervals are shown in Figure 2.

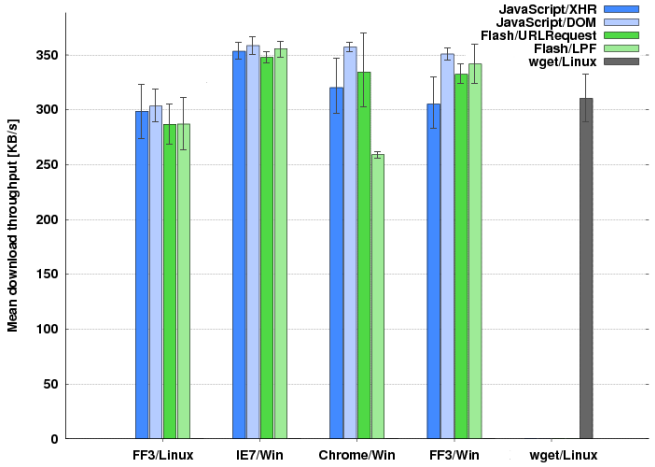

Figure 2. Means and Confidence Intervals for Controlled Download Throughput

Examination of the individual results indicates that, while there is variability within the results for each method, the download throughput is similar across methods. It is also noteworthy that the implicit techniques do not show significant differences from the explicit technique nor from the baseline `wget` measurements. These results suggest that download throughput to third-party hosts can be determined as accurately as to the origin server.

Upload throughput measurements were done by using the two explicit techniques (XMLHTTPRequest and URLRequest) to send a request for a small (1 byte) file with 2MB of POST data in each of the four tested browsers. Validation was done by using `wget` to request the same small resource and using the `--post-file` option to attach a 2MB file as a POST variable. The results for each method and their confidence intervals are presented in Figure 3.

The results, while consistent for each technique, show significant differences between measurement techniques. Confirmation that the duration of the upload is consistent

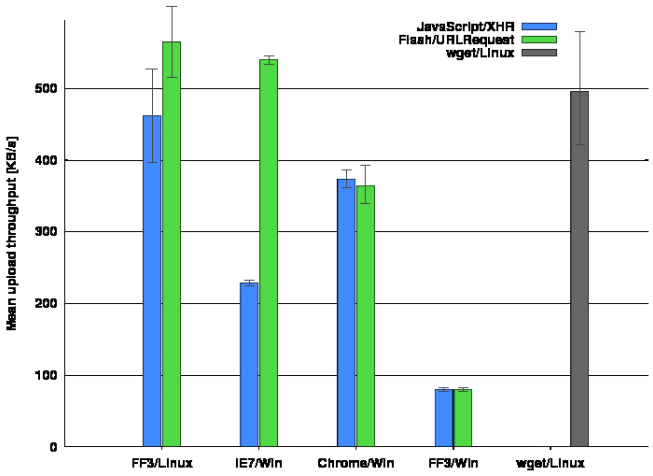

Figure 3. Means and Confidence Intervals for Controlled Upload Throughput

with data obtained from the network layer ruled out errors in our techniques or overhead in the browser. Uploads from the Linux platform use a TCP window size of 128KB while uploads from Windows use a window size of 32KB. This difference helps explains the difference in results for Firefox in Figure 3, but not differences between browsers on the same Windows OS. Ongoing work is to better understand these differences.

Round-trip time was measured using the four developed mechanisms in each tested browser by requesting a 1 byte file from the Web server. The baseline test used the `tcpping` program from the Linux client to determine the time necessary to establish a TCP connection with the server. The mean results and confidence intervals are given in Figure 4.

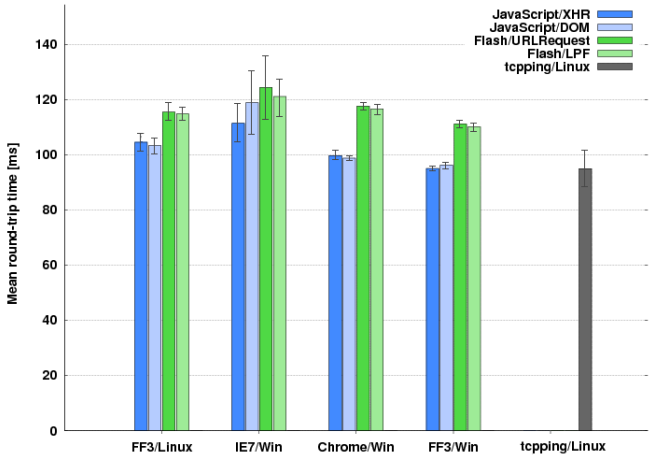

Figure 4. Means and Confidence Intervals for Controlled RTT

As with download throughput results, the mean values obtained for most measurements techniques fall within a narrow range. The RTT measured using `tcpping` shows,

predictably, the lowest RTT to the server, as it only measures the time to establish a connection and does not request any resources from the Web server. An important observation is that Flash techniques report, on average, higher RTTs than JavaScript techniques. However, those results are still within a 20% margin (about 25ms) from the minimum RTT reported by `tcpping`. The results indicate that all techniques are generally useful for determining the RTT.

4.2 Open Tests

After establishing the correspondence between throughput and RTT metrics obtained using the developed browser techniques and results obtained via standard tools, a larger-scale study was conducted to determine relative differences between the measurement methods themselves over a range of client environments.

Users were invited to participate in the study with volunteers both accessing the test server via the WPI network as well as various Internet Service Providers (ISPs). For each client, the tests measured download and upload throughput to the test server, RTT to the server, and RTT to selected third-party hosts using each of the four implemented techniques. Once started, tests were repeated every five minutes until the user decided to close the browser window containing the test page. Incomplete test results (with less than one full execution of the test suite) were discarded. The analyzed results are based on data from about 20 users on the WPI network and 15 non-WPI broadband users of whom 13 connected from a known major broadband ISP (Verizon, Charter, Comcast or Covad). A total of 3610 individual measurements were obtained from WPI-based users (1441 download, 731 upload, and 1438 RTT measurements). Broadband users performed 3393 tests (1390 download, 660 upload, and 1343 RTT measurements). The results presented are for broadband users only, with complete results for all users available in [5].

In analyzing results we examined the correlation of the four techniques across the set of measurements. Figure 5 shows the comparison of all download throughput values obtained with both JavaScript and Flash with a best fit line having a slope of 1.02 +/- 0.07 (95% confidence interval). These results are also broken down to compare the download throughput for explicit and implicit techniques in JavaScript and Flash. For JavaScript the slopes are 1.04 +/- 0.10 and for Flash the slopes are 1.05 +/- 0.10, indicating that there are no systematic differences between explicit and implicit techniques for broadband download throughput. Similar to controlled testing, the open client results indicate that for broadband clients there are no significant differences in the average download throughputs between the techniques.

The upload throughput comparison for the XMLHTTPRequest and URLRequest explicit techniques is shown in Figure 6. The results indicate that for broadband clients, URLRequest upload reports slightly higher upload rates than XMLHTTPRequest (1.13 +/- 0.08).

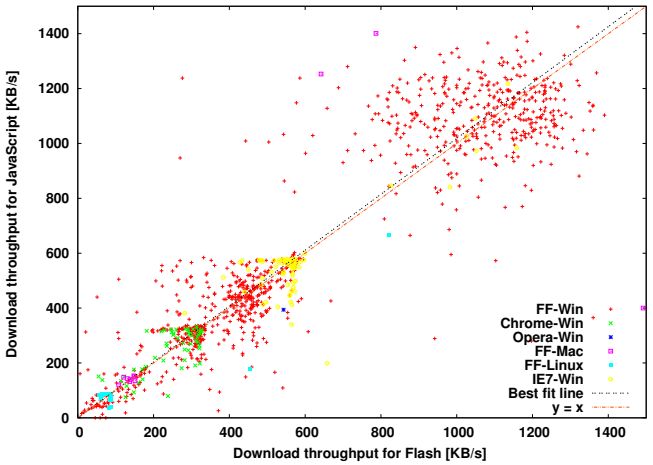

Figure 5. Comparison of Download Throughput Measured by JavaScript and Flash Methods for Broadband Connections

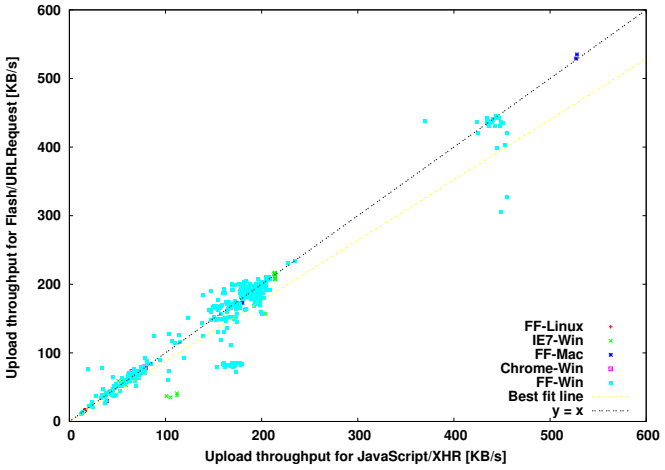

Figure 6. Comparison of Upload Throughput Measured by XHR and URLRequest for Broadband Connections

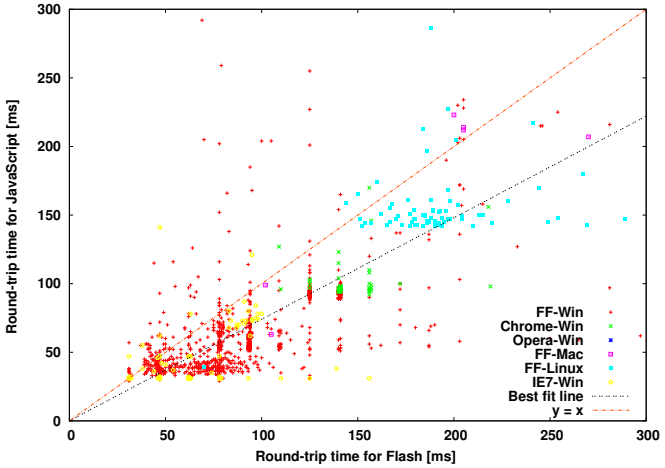

Figure 7. Comparison of RTT Throughput Measured by JavaScript and Flash Methods for Broadband Connections

Comparisons between explicit and implicit methods within Flash and JavaScript for measuring RTT indicate that is no significant difference between the mechanisms. However, Figure 7 shows there exists a large difference between average RTT results as reported by JavaScript and Flash. RTT results reported by JavaScript are on average almost 30% lower than by Flash techniques with a best-fit slope of 0.74 +/- 0.06.

As part of the open study, RTT was analyzed to three third-party servers: `boston.com`, `unl.edu` and `youtube.com`. These three servers were selected because of their geographic location (U.S. East Coast, U.S. Midwest and U.S. West Coast), to provide different latency conditions for test clients, most of which connected from the East Coast. For this experiment results were gathered only using implicit techniques as the same-origin policy prohibits making explicit requests to third-party hosts.

Correlation of results for broadband users using the JavaScript and Flash implicit techniques shows the DOM technique reports smaller round-trip times for `boston.com` (slope 0.71 +/- 0.10) as shown in Figure 8. We found similar results for `unl.edu` (slope 0.76 +/- 0.08) and `youtube.com` (slope 0.83 +/- 0.08). We suspect that a large part of the discrepancy is caused by a constant overhead inherent in the Flash LPF technique. The effect of this overhead lessens as the RTT increases.

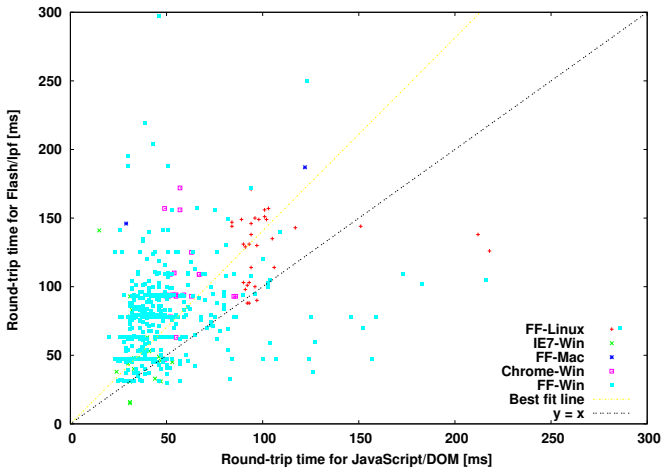

Figure 8. Round-Trip Time to `boston.com` for Broadband Hosts

4.3 Jitter

Jitter results were analyzed for streaming data from the test server at WPI to our controlled client connecting from a campus residential network on the U.S. West Coast. A 10KB file was divided into 40 chunks of 250 bytes, spaced at 100ms. The transfer was performed using the Flash URLRequest technique under conditions with no interference and with a competing concurrent file download from the origin server.

As the results in Figure 9 show for the no-interference scenario, there is almost no variation in the reception times for all packets, indicating *good* network conditions. On the other hand, as shown in Figure 10, the results with interference show that about 30% of packets were delayed, with 15% delayed by over 50ms.

The specific results here are not necessarily important, but they show that the jitter measurement technique has potential for evaluating network conditions outside of the often-measured throughput and RTT metrics; the technique also provides an opportunity to detect transient network interruptions and give insight into the performance of streaming data within the Web browser environment.

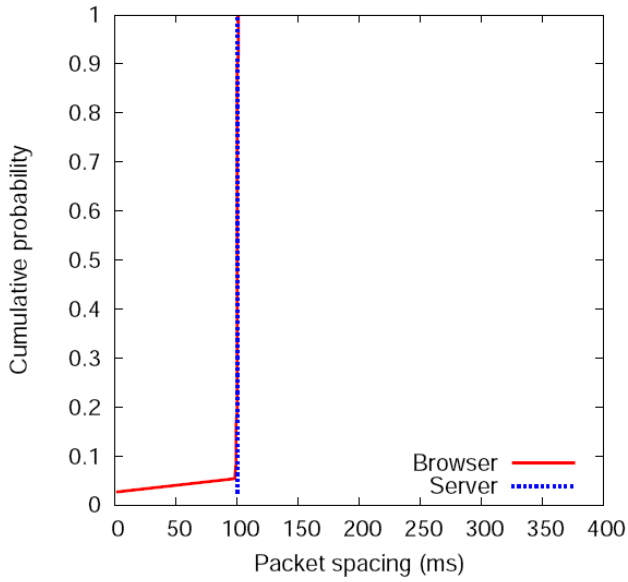

Figure 9. Packet Streaming without Interference

5 Conclusions

In summary, we developed new implicit techniques for the measurement of download throughput and RTT that can be used between a client and arbitrary Web servers on the Internet. We found that these implicit techniques perform on par with explicit techniques. This result suggests that the JavaScript DOM and Flash LPF techniques can be used to reliably estimate client download throughput and RTTs to arbitrary third-party hosts, which could lead to customized tests and reports specifically for multimedia, game and other application sites of interest to users. We also developed a technique for measuring delay jitter over TCP within the browser and showed that to be effective.

We were also able to compare the performance of measurement techniques across a variety of operating system and browser environments. We found the throughput and RTT measurements within 20% of expected baseline values. The developed techniques provide significant variation in upload throughputs between browsers on the same

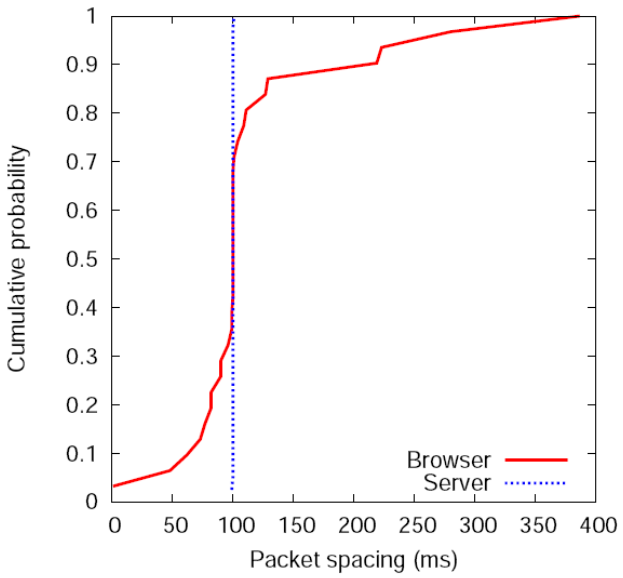

Figure 10. Packet Streaming with Interference

client host, indicating that they are not generally sufficient to determine a connecting client's network upload bandwidth.

For broadband users, all developed techniques perform equally well in the case of upload and download using the same browser. The average RTT measured in Flash is higher than the RTT measured in JavaScript.

There are a number of directions for future work. Further measurements of the techniques with more clients would provide even stronger validation and may yield a better understanding of differences in the upload throughputs. Longer term, this work can be built upon to provide an interface and results that are more meaningful to users with the use of games and videos to attract users as well as provide feedback and visualization for retaining them.

References

[1] Adobe. ActionScript 3.0 Language and Components Reference: URLLoader, Jan 2009. `http://livedocs.adobe.com/flash/9.0/ActionScriptLangRefV3/flash/net/URLLoader.html`.

[2] Adobe Systems Incorporated. Flash Player Version Penetration, Sep 2008. `http://www.adobe.com/products/player_census/flashplayer/version_penetration.html`.

[3] Broadband reports.com speed test. `http://www.dslreports.com/stest`.

[4] A. Janc. nettest. `http://code.google.com/p/nettest/`.

[5] A. Janc. Network measurement using Web sandbox environments. Master's thesis, Computer Science Dept., Worcester Polytechnic Institute, January 2009.

[6] Ookla. Speedtest.net - The Global Broadband Speed Test, Jan 2009. `http://speedtest.net`.

[7] L. Peterson, A. Bavier, M. E. Fiuczynski, and S. Muir. Experiences building PlanetLab. In *Proceedings of the 7th USENIX Symposium on Operating Systems Design and Implementation*, Seattle, WA USA, November 2006.

[8] A. Ritacco, C. E. Wills, and M. Claypool. How's my network? a Java approach to home network measurement. In *Proceedings of the IEEE International Conference on Computer Communications and Networks*, San Francisco, CA USA, August 2009.

[9] J. Ruderman. The same origin policy, 2001. `http://www.mozilla.org/projects/security/components/same-origin.html`.

[10] Y. Shavitt and E. Shir. DIMES: let the Internet measure itself, 2005. `http://www.arxiv.org/abs/cs/0506099v1`.

[11] C. R. Simpson, Jr., D. Reddy, and G. F. Riley. Empirical models of TCP and UDP end-user network traffic from NETI@home data analysis. In *20th ACM/IEEE/SCS Workshop on Principles of Advanced and Distributed Simulation (PADS 2006)*, pages 166–174, May 2006.

[12] W3 Schools: Browser Statistics, Jan 2008. `http://www.w3schools.com/browsers/browsers_stats.asp`.

[13] W3C. The XMLHttpRequest Object, Apr 2008. `http://www.w3.org/TR/XMLHttpRequest/`.

[14] Z. Wen, S. Triukose, and M. Rabinovich. Facilitating Focused Internet Measurements. In *SIGMETRICS: Proceedings of the ACM International Conference on Measurement and Modeling of Computer Systems*, New York, NY, USA, June 2007. ACM Press.