



Microsoft

Measuring Browser Performance

Understanding issues in benchmarking and performance analysis

Published March 2009

For the latest information, see <http://www.microsoft.com/ie8>

Version 1.0

Abstract

This document will explain various browser components and how each can impact performance when benchmarking. The abilities and pitfalls of various benchmarking tools will be compared, and ways to design tests that avoid these issues will be discussed. Also included is an overview of instructions on how to set up a benchmarking environment to conduct some of the testing processes discussed here.

The information contained in this document represents the current view of Microsoft Corp. on the issues discussed as of the date of publication. Because Microsoft must respond to changing market conditions, it should not be interpreted to be a commitment on the part of Microsoft, and Microsoft cannot guarantee the accuracy of any information presented after the date of publication.

This guide is for informational purposes only. MICROSOFT MAKES NO WARRANTIES, EXPRESS OR IMPLIED, IN THIS SUMMARY.

Complying with all applicable copyright laws is the responsibility of the user. Without limiting the rights under copyright, no part of this document may be reproduced, stored in or introduced into a retrieval system, or transmitted in any form, by any means (electronic, mechanical, photocopying, recording or otherwise), or for any purpose, without the express written permission of Microsoft.

Microsoft may have patents, patent applications, trademarks, copyrights or other intellectual property rights covering subject matter in this document. Except as expressly provided in any written license agreement from Microsoft, the furnishing of this document does not give you any license to these patents, trademarks, copyrights or other intellectual property.

Unless otherwise noted, the example companies, organizations, products, domain names, e-mail addresses, logos, people, places and events depicted herein are fictitious, and no association with any real company, organization, product, domain name, e-mail address, logo, person, place or event is intended or should be inferred.

Microsoft, Windows, SmartScreen, InPrivate and Internet Explorer are either registered trademarks or trademarks of Microsoft Corp. in the United States and/or other countries. The names of actual companies and products mentioned herein may be the trademarks of their respective owners.

© 2009 Microsoft Corp. All rights reserved.

Table of Contents

Measuring browser performance	4
Testing for the real world.....	4
Testing tools.....	5
Caching behavior	5
Measurement overhead and resource competition.....	6
Connectivity and network device latency	6
Dealing with Internet-based data	6
Browser targeted content	7
Machine Configurations	7
Extensibility issues	8
Inconsistent definitions.....	9
Measuring for yourself.....	9
Step 1. Choose testing targets	10
Step 2. Setting up the system.....	10
Step 3. Test in sequence.....	10
Step 4. Warm start testing	10
Step 5. Do the benchmarking	10
Step 6. Repeat.....	11
Putting it all together	11
Appendix A: List of Sites Tested	13

Measuring browser performance

Users arguably spend more time using their web browser than any other application on their machine, so browser vendors must work hard to make their products stable, robust and fast. Benchmarking browser speed is easier said than done. This document will explain various browser components and how each can impact performance when benchmarking. Benchmarking tools will be compared, and we will present ways to design tests that avoid these issues. Lastly, this whitepaper includes an overview of how to set up a benchmarking environment that will enable you to conduct some of the testing processes discussed here. To go directly to the step-by-step instructions on how to conduct a benchmarking test, jump ahead to the section titled “Measuring for Yourself.”

In the Internet Explorer testing lab, we run daily and monthly benchmarking passes in our controlled testing environment using a tiered testing model to catch regressions in the product. This process helps ensure that we are able to get a regular and consistent view of performance, as well as providing the ability to scale and test tens of thousands of websites. The daily testing covers approximately 25 websites, such as those listed in Appendix A, although the specific list varies from day to day. Websites used in daily testing are chosen for a variety of reasons, but one important factor included in selection criteria is that the list incorporates a wide variety of websites, including international websites, in order to be representative of users on the Internet today. For the Internet Explorer 8 performance video we have published, we chose the Top 25 websites in the world as measured by the Global Internet Information provider comScore to reflect the sites users are most likely to visit on the web.

Testing for the real world

It is difficult to measure the performance of a browser by using “live” websites. There are many variables such as network latency, network congestion, and website traffic levels for “live” website tests that can affect measurements. To address this, artificial lab tests (benchmarks) are created to measure specific aspects of browser performance in a controlled setting. These benchmarks necessarily characterize only a narrow set of the browser functions in a very constrained way. End users, however, do not operate in a controlled environment, nor do they perform only a subset of their browsing tasks. When end users use their browser, they perform complete tasks, such as reading a news story, getting directions or online shopping.

Browsers are composed of several subsystems, each of which plays a role in overall performance. End users don’t just run JavaScript or load test pages, so testing “micro-benchmarks” does not translate directly to the end user experience and can be a misleading indicator of overall performance. End users are more properly aligned with tests that measure overall “page load times” which represent the completion of a task, not isolated operations.

Think about a real world task like going to the grocery store. The task is filled with many smaller functions, each critical to the task’s overall completion, but speed in one step does not correctly indicate the entire process will be fast. For example, imagine two neighbors who have the same grocery list. If one shopper leaves for the store in a station wagon at the same time his neighbor leaves in a sports car, it doesn’t mean the neighbor in the faster car will finish shopping first. Once at the store, the neighbor may be less efficient at navigating the store aisles and lose time finding items on his list. At checkout,

one shopper may use cash, while the other pays by check, which also takes longer. As this example illustrates, measuring only one portion of the process can easily offer misleading direction on the overall performance.

For the real world user, performance criteria are based on how quickly a page loads, not the performance of any individual subsystem. For that reason, this paper will focus on how to measure complete webpage load times and discuss the related complexities. We will discuss the testing tools and their limitations, the affects of caching, the impact of measurement overhead and resource competition, and the variance introduced by network connectivity and other resource limitations. We will describe each of these and look at how and why they affect performance testing relative to the goal of measuring page load times.

Testing tools

Many benchmarking tests cite results from tools such as SunSpider, Celtic Kane, V8 and iBench. Each is designed, at least in some way, to test only a portion of the end-user scenario. SunSpider primarily tests JavaScript performance and also some other popular functions used on websites today. Other popular benchmarks, Celtic Kane and V8, are also geared at testing only the JavaScript engine. Meanwhile, the iBench suite relies on a browser reporting when it's finished loading a page. Since browsers differ on when they define a page is done loading, this test also yields unreliable results.

Caching behavior

Caching, a common design feature used by browsers to optimize the users online experience and can impact benchmark testing. Browsers are designed to cache (store) content locally in order to avoid repeatedly requesting the same content. Many company networks and Internet Service Providers (ISPs) similarly exploit caching to greatly reduce the amount of content repeatedly requested over their network transmission lines, so users do not suffer delays when browsing a site with multiple pages containing the same content, such as company logos or other common graphics.

However, this timesaving design can introduce factors that incorrectly impact benchmarking results. For example, if a test calls for opening 10 tabs to 10 different websites in one browser, and then to open the same 10 tabs in a second browser, the results could wrongfully indicate that the second browser was faster, when in fact the difference was due primarily to the content being stored by a nearby server when the first browser requested the pages.

In the Internet Explorer lab: We visit each site prior to starting any site test. "Preloading" the cache prior to a test helps ensure systems are at a known base before starting.

For you to try: Because it's nearly impossible to avoid all network-level caching devices, one solution is to visit each Web site you plan to test—with each browser you plan to test—before starting the actual test. This "pre-caching" step helps ensure that each browser is being tested more equally.

Measurement overhead and resource competition

Another benchmarking problem is that the process of measuring consumes resources and can impact the application you are trying to measure. This process is known as the “observer effect.” For example, adding debug code to an application may cause the program to execute more slowly than normal. There is a related condition, known often as a “Heisenbug,” in which the act of changing the code to measure it (or isolate a condition for bug testing) ends up altering the effect of variables or other system activities.

Because there is no way to completely remove the measurement process load overhead, the goal is to find ways to minimize it. A popular approach is to use tests that focus on specific scenarios. The theory is that, while still having some performance impact, smaller tests yield fewer extraneous activities.

Perhaps the largest issue in benchmarking is resource competition. Any given system only has so many resources, and they cannot all be available to every application at the same time. A process cannot start until resources are available, which could result in inconsistent benchmarking results, as resources are constantly in flux. Other applications running on the machine are competing for resources, so efforts should be taken to isolate or reduce them.

In the Internet Explorer lab: We disable all other unnecessary applications and services.

For you to try: To help reach a valid configuration, users should disable other applications during testing.

Connectivity and network device latency

Another issue is that while network connectivity is constantly increasing, network resources are not unlimited. Several things, like local area congestion, trunk line congestion and peering congestion, can impact network resource availability. Applications running at the network layer (such as proxy servers, load balancers, firewalls, and so on) can also have resource contention issues that impact benchmarking data.

In the Internet Explorer lab: We use a DTAP in our testing to mitigate many of these issues with our testing network.

For you to try: Because there is no way to completely isolate or remove these issues, the best way to address this is to identify common load periods and test under similar load levels each time.

Dealing with Internet-based data

Testing on Internet-based (“live”) Web servers can introduce factors that impact performance data observations. Simply put, what the server is doing and how the application is designed can impact the benchmarking process. Fortunately, many application issues can be isolated or removed entirely. Server load, however, can’t be dealt with in the same way.

In the Internet Explorer lab: We perform several rounds of tests over time and remove outliers to get a statistical average.

For you to try: The best solution to counteract these issues is to conduct several tests over a period of time and ensure consistency in network layer resources during those tests.

Browser targeted content

Many websites perform browser detection in order to provide users with the best possible experience when using their chosen browser. This can result in a website behaving differently when viewed with one browser versus another, but it may also cause the website to serve varied content depending on the client browser. That variation can lead to benchmarking errors and irregularities when content viewed in one browser cannot be directly compared to another. Slight differences in CSS or layout designs are generally acceptable—and often unavoidable—but large differences should be avoided. It is important to visually review each website prior to benchmark testing to verify the website isn't drastically different when viewed with different browsers.

In the Internet Explorer lab: We review each website on the list in each browser we are testing prior to starting a test pass. Any websites which appear drastically different are removed from the test list to help ensure a more consistent apples-to-apples comparison.

For you to try: Make a list of websites to be used in benchmark testing. Visit each one using each browser to be tested and visually inspect them to ensure no large differences between the appearance or layout of each website.

Machine Configurations

Just as with humans, no two machines are exactly alike. As a case in point, there are several dozen machines in the Internet Explorer performance lab running performance tests every hour of every day. To maximize our lab's flexibility, an attempt was made to create a set of "identical" machines which could be used interchangeably to produce a consistent set of performance data. Those machines bore consecutive serial numbers and were from the same assembly line, and all their component parts were "identical."

To understand this, it is important to remember that hardware vendors obtain parts from a broad range of suppliers in an effort to ensure sufficient supplies as well as manage costs. One common component, for example Ethernet ports, may be sourced from as many as three suppliers. As long as each product meets the proper engineering specifications, those parts can be used interchangeably—an Ethernet port is an Ethernet port, much like a Cat 5 cable can be interchanged with any other Cat 5 cable. As long as both meet the prescribed engineering standards, they should function in the same way. The problem is that minor differences such as device drivers can introduce performance issues of their own. Since there is no easy and scalable way to ensure all system components are truly identical, the best course is to minimize comparing test results from different machines.

For these reasons, despite all the efforts to have duplicate systems, data collected on the machines in the Internet Explorer performance lab has been sufficiently varied that it is best not to compare performance results from two different machines.

In the Internet Explorer lab: Prior to starting each benchmarking test cycle, machines are re-imaged to help ensure the most pristine state possible.

For you to try: Set up a machine for your benchmarking tests, free of other applications and default settings that may impact performance. Use that same machine for all testing to ensure consistency.

Extensibility issues

The issue of extensibility, or add-ons, is yet another potentially confounding element when benchmarking. Though browser add-ons offer users a powerful way to get more from their browsing experience, they can also introduce performance issues, making it difficult to distinguish an “original” browser issue from one introduced by the third-party code. Mike Shaver, Chief Evangelist at Mozilla, noted this issue in an interview: “People using Firefox sometimes load it up with a pile of extensions and then saying, ‘Performance isn’t what it used to be’ or ‘I get this one weird crash.’”¹

To account for this, it is important to test both “clean” installations without any add-ons, as well as some with popular add-ons enabled (such as Adobe Flash, Microsoft Silverlight®, and Windows Media® Player). Comparing the results of those two data sets should help give some visibility into the impact of third-party elements.

To disable add-ons for Internet Explorer 8, select **Manage add-ons** from the **Tools** menu. In the **Manage add-ons** dialog box, ensure that **All add-ons** are visible, and then disable all add-ons (this can also be done at a command prompt by typing `iexplore.exe -extoff`). Alternatively, can be run in “Safe Mode,” in which the browser is launched without any add-ons enabled. This mode can be found by going to **Accessories > System Tools**. Some browsers offer a form of safe mode operation, making it easier to benchmark without add-ons across different browsers.

<i>Browser</i>	<i>To Enable Safe Mode</i>
Internet Explorer	Start > All Programs > Accessories > System Tools > Internet Explorer (No Add-ons)
Firefox	Start > Mozilla Firefox > Mozilla Firefox (Safe Mode)

In the Internet Explorer lab: We install and test with the most common add-ons (Flash, Silverlight and Windows Media Player) necessary for common functionality on popular websites

¹ [Scott Swigart](http://howsoftwareisbuilt.com/2008/03/20/interview-with-mike-shaver-chief-evangelist-mozilla/) and [Sean Campbell](http://howsoftwareisbuilt.com/2008/03/20/interview-with-mike-shaver-chief-evangelist-mozilla/). How Software Is Built. 20 March 2008.
<http://howsoftwareisbuilt.com/2008/03/20/interview-with-mike-shaver-chief-evangelist-mozilla/>

to ensure a real world user experience. We also test with no add-ons installed in order to get side-by-side measurement data.

For you to try: Take an inventory of the add-ons installed on the browsers and ensure they are the same; then disable or remove any add-ons which are not available (or not equal) to those available for other browsers you plan to test. In addition, tests should be run with no add-ons to get comparison benchmarking data.

Inconsistent definitions

Another factor that impacts benchmarking is having a consistent baseline of what it means to complete a task. In car and horse racing, the end is clear—there is a finish line that all contestants must cross to finish the race. Browser benchmarking needs a similar way to define when a webpage is done loading. The problem is that there is no standard which dictates when a browser can or should report that it is “done” loading a page—each browser operates in a different way, so relying on this marker could produce highly variable results.

The best approach is to utilize browser progress indicators and validate those indicators against when the webpage is actually loaded and ready for use. For example, when testing how quickly a particular webpage loads, look for a specific object or try to interact with the page while it is loading for the first time. If the webpage appears to be loaded and is interactive before the progress indicators complete, it is reasonable to ignore the indicators and use the page appearance for measurements. If the webpage is not ready for use, then the progress indicators should be sufficient to make an initial assessment of how quickly a page is downloading across various browsers.

In the Internet Explorer lab: To avoid the inconsistent results of using the “Done” status notification, we primarily use visual cues to determine when a page is complete. For sites on which specific visual cues aren’t sufficient, we use a combination of visual cues and the ability to interact with content on a page to determine if a page is “done.” Some site designs utilize AJAX or similar technologies where determining “done” is even more complex, so in those cases it is more deterministic to use a visual cue.

For you to try: Avoid using any browser self reporting status information and rely on your own experiences and responses. If you are able to interact with the page and it appears ready to use, it is acceptable to use that timing data. In cases where the site design is overly simplified (for example, where only a few objects are on a page) and it would be difficult to measure when the page is ready for use, you may find it easier to look for a visual property on the page and count the page as complete when that item is loaded.

Measuring for yourself

Armed with an understanding of the issues involved with benchmarking and ways to avoid those pitfalls in testing, the next step is to build a benchmarking test bed.

In the Internet Explorer lab: We use a video recording system to monitor all tests, which enables us to slow down the video for a frame-by-frame replay to capture extremely accurate performance times and read measurement cues.

For you to try: While it is impractical to provide a blueprint to test as the Internet Explorer lab does, these steps will provide a basic environment for completing a suitable evaluation.

Step 1. Choose testing targets

Vary testing selection. Having a list of popular Web sites is important, but if the sites are too similar in content and functionality, you will not fully test the browser's subsystems. Having a broad and mixed set of testing targets helps ensure you test all of the different browser subsystems rather than just one as many micro-benchmarks like SunSpider do. The list of sites we used is included in Appendix A.

For each testing target, ensure user interaction steps are clear and understood before continuing. During the actual testing process, it is necessary to understand the testing target well enough to determine whether the site is usable; that is, that a user can navigate and interact with the site. If the test is going to be based on specific visual cue items, ensure those items are well known and recognizable.

Step 2. Setting up the system

Clear the cache. It is vital to ensure each browser is in a clean state before testing starts. Each browser has a slightly different way to clear the cache. For Internet Explorer, go the **Safety** menu and select **Delete Browsing History**. Since this feature is designed to delete several items in one menu, ensure that only the **Temporary Internet Files** option is selected. This will ensure each browser is in a common state.

Stop other applications. To do this, close other open applications and look at applications running in the system tray. Where possible, close anything in the system tray. Launch **Windows Task Manager** (by pressing CTRL+SHIFT+ESC), highlight any tasks in the **Applications** tab (including the browser being tested) and select **End Task**.

Step 3. Test in sequence

Select one browser. It is important not to test side-by-side: that is, having both browsers running at the same time and accessing the target website. Doing so can impact one or both browsers and skew results.

Step 4. Warm-start testing

Load the browser. Launch one browser and visit the target website(s) selected for testing to prepare the cache and get various browser components loaded. This ensures there are no extra delays or performance impacts introduced by loading pieces. Wait to warm-start the other browser(s) being benchmarked until the testing cycle on the previous browser is complete.

Step 5. Do the benchmarking

Set up a video recording device. Capturing accurate timing information is critical for performing a detailed browser performance comparison. Using an external video recording device, either by capturing the PC video output or by using a standalone camcorder, avoids any impact or system

overhead from affecting benchmarking results and enables you to slow down footage later to compare results. The Internet Explorer team captured video directly from the PC output to document benchmark timings. Set up the device in a location with a clear view of the screen and ensure the framing of the screen is cropped to fill the whole frame. Ensuring the video image is as large and clear as possible will help you identify even small objects as they load on the screen. Once the video capture is complete, you can replay the video by using media player software to slow down the images to get granular timing information. The Internet Explorer team uses media player software capable of viewing data frame-by-frame², providing the ability to obtain timing data at the accuracy of approximately 1/30th of a second. To enable frame-by-frame playback using Windows Media Player, select the **View** menu, then **Enhancements**, then **Play Speed Settings**. This will display a controller which allows you to advance by single frames. Other media player software applications also support frame-by-frame playback so you may be able to use your preferred software in place of Windows Media Player.

Another option for timing: get a stopwatch. If you are unable to set up a video recording device, it is possible to use a stopwatch to capture timing data. However, the stopwatch method introduces the potential for timing delays and inaccuracies, meaning the results are much less reliable and so should be avoided if at all possible. The stopwatch method creates a small delay, known as the “Personal Equation” (PE), which is the reaction time between the eye first noticing an event and the hand clicking the stopwatch. The PE for most people is around 0.4 seconds, so that amount should be removed from each start and stop measurement.

Start the testing. Using your list of websites, enter the URL into the browser’s Address bar. Prepare the stopwatch and start timing when hitting ENTER. To avoid inconsistency in how each browser reports when a webpage is complete, record “done” times as the moment the visual cue appears or when the user is able to interact with the site. For example, if the site has a text entry box, “done” would mean when the user can enter text.

Get multiple measurements. Since both internal and external factors can impact time measurement, it is important to conduct 10 tests for each Web site on the list. Once the list is complete, remove the top and bottom 2 samples and average the remaining numbers.

Step 6. Repeat

Test other browsers. Repeat Steps 4 and 5 for each browser being tested.

Test again later. To avoid any issues with time of day network traffic and utilization impacts, all tests should be repeated at another time of day. Ideally, one set of tests would be done during normal working hours and the second approximately 12 hours later. Normal working hours should correspond to the target user base for the given website.

Putting it all together

As this analysis has illustrated, there are many factors involved in benchmarking and many steps involved in attaining valid results. This document should provide the information needed to perform benchmarking tests and isolate extraneous factors to allow for a true analysis of browser performance.

² NTSC video records at 30 frames per second (fps).

Having an understanding of the limitations of popular benchmarking tools will help everyone understand why it is important to view performance on a holistic level, looking at the bigger picture of page load times, which account for all browser subsystem elements.

Appendix A: Results from Browser Performance Test

For the Internet Explorer 8 performance video we published, we chose the Top 25 websites in the world as measured by [comScore](#) to reflect the sites users are most likely to visit on the web. The following table shows results from the benchmark testing process for those 25 websites tested on the following browsers: Google Chrome, Mozilla Firefox, and Internet Explorer 8.

As noted earlier, Microsoft chooses approximately 25 websites for daily testing, and tens of thousands on a monthly basis. Any list of websites to be used for benchmarking must contain a variety of websites, including international websites, to help ensure a complete picture of performance as users would experience on the Internet.

Summary of Browser Load Times (in seconds)				
Rank	Web Site	Chrome 1.0	Firefox 3.05	Internet Explorer 8
1	google.com	0.28	0.22	0.20
2	yahoo.com	2.15	1.30	1.32
3	live.com	3.48	3.42	3.40
4	msn.com	0.55	0.97	0.83
5	youtube.com	3.07	2.80	2.55
6	microsoft.com	3.83	3.47	3.75
7	wikipedia.org	1.22	1.35	1.88
8	blogger.com	2.07	2.88	1.52
9	facebook.com	2.12	2.08	1.98
10	qq.com	6.82	7.88	7.33
11	baidu.com	1.40	1.47	1.50
12	myspace.com	2.13	5.53	3.68
13	wordpress.com	0.95	1.58	1.45
14	ebay.com	4.06	4.43	3.42
15	sina.com.cn	5.48	6.37	8.03
16	mozilla.com	1.28	1.53	1.27
17	adobe.com	9.50	9.37	8.85
18	aol.com	3.02	2.57	2.32
19	amazon.com	2.12	3.35	2.82
20	apple.com	2.52	3.07	1.63
21	soso.com	3.25	2.64	2.07
22	xunlei.com	8.60	9.97	8.70
23	163.com	14.87	14.75	15.02
24	google.cn	1.38	0.85	1.05
25	ask.com	2.17	1.77	1.73

In line with the testing guidelines outlined in this paper we have used the browser “Done” indicator for timing when the page when the page is completely loaded at that point. For pages which continue to load and change after the “Done” indication we have used common visual cues to generate the timings. Timing is started when the **Go** button is pressed. These timings were captured in January 2009; because Internet content is always changing you may get different timings when you run these tests.