End-to-End Throughput and Response Time Testing
With HTTP User Agents and the JavaScript Language
<draft-rfced-info-stanislevic-00.txt>

Status of this Memo

This document is an Internet-Draft. Internet-Drafts are working
documents of the Internet Engineering Task Force (IETF), its areas,
and its working groups. Note that other groups may also distribute
working documents as Internet-Drafts.

Internet-Drafts are draft documents valid for a maximum of six months
and may be updated, replaced, or obsoleted by other documents at any
time. It is inappropriate to use Internet-Drafts as reference material
or to cite them other than as "work in progress".

To view the entire list of current Internet-Drafts, please check the
"iid-abstracts.txt" listing contained in the Internet-Drafts Shadow
Directories on ftp.is.co.za (Africa), ftp.nordu.net (Europe),
munnari.oz.au (Pacific Rim), ds.internic.net (US East Coast), or
ftp.isi.edu (US West Coast).

Abstract

This memo describes two simple metrics and a methodology for testing
end-to-end one-way data throughput and two-way response time at the
application layer utilizing HTTP [1] (web) servers and user agents
(web browsers). Two Interactive Hypertext Transfer Test (IHTTT)
implementations are described in detail.

Acknowledgments

This memo and in particular, Section 2c, were inspired by the work of
the Internet Protocol Performance Metrics (IPPM) IETF working group.

Table of Contents

1. Introduction ........................ 2
1a. Interest ............................ 2
1b. Motivation .......................... 3
2. Discussion ............................ 4
2a. Advantages .......................... 4
2b. Caveats .............................. 5
2c. Statistical Sampling ................. 6
3. Metrics ............................... 9
3a. User Data Throughput ............... 9
3b. User Response Time .................. 9
3c. Combined User Response Time/Data Throughput .... 9
3d. Some Other Interesting Derived Metrics .... 10
4. Implementations of Test Methodologies .... 10
4a. Test Launch Page .................... 11
4b. User Response Time Page ............. 12
4c. Combined Test Page ................... 12
5. Test File Names ....................... 12
1. Introduction

1a. Interest

In the absence of sophisticated tools and methodologies to measure application layer data throughput and response time via complex network topologies, simple file copy tests are often performed by end users and network designers alike. The scope of such tests encompasses not only network layer entities (e.g. routers, links and clouds), but also client and server hosts. These tests are often performed manually using a variety of sample files. Typically, the time taken for a given size file to be transferred from a server to a client is measured. The file size (in bits) is then divided by the measured time (in seconds), yielding a throughput rate in bits per second (Bytes per second * 8). Separately, or in conjunction with these tests, the time required to request, transfer and display a small amount (e.g. one line) of test data is measured. The former test can be said to measure one-way application layer, (or "User") Data Throughput and the latter, two-way User Response Time.

This memo describes automated versions of the above tests which can reside on any web server, and be easily performed by any end user. The objective is to allow end users to run these types of tests via HTTP [1] connections in real time with web browsers, thereby obtaining useful end-to-end performance data without the need for additional tools.

To achieve the above objective:

- All client software shall be contained within the user agent (web browser);

- All test data samples, measurement indicators and measurement applications shall be contained in HTML [2] files on the web server;

- All measurements shall be performed by the client, using its internal clock. (A single clock source is always self-synchronized, thereby exhibiting little relative skew or drift. For this reason, external time standards are not required.);

- All test results shall be collected and displayed by the client in real time and shall be exportable to analysis tools (e.g. spreadsheets).

As the test methodology in this memo resides at the application layer, its use is not limited to HTTP connections. It will work via connections established using any file copy protocol capable of transporting HTML. However, to be most relevant within the context of
the Internet, we will limit the scope of our discussion to HTTP over wide-area networks.

It is intended for this memo to stimulate discussion, leading eventually to the adoption of standards and the proliferation of these, or other similar, test files on many sites around the Internet. With the above web sites as sources of the test data and measurement applications, basic real-time application layer performance tests could be carried out by end users at any time, simply by browsing the appropriate web pages.

1b. Motivation

(1) HTTP and World Wide Web services have become ubiquitous on the Internet. It is now possible to access information on nearly any subject in real time (within the bounds of network, client and server performance) using this protocol. For the average user, however, real-time and cumulative information about the performance of particular HTTP connections is harder to come by. Experience has shown a great deal of variation in user-perceived performance from site to site, time to time and ISP to ISP. Work is in progress by the IETF Internet Protocol Performance Metric working group to develop Internet Provider Performance Metrics for both connection oriented and connectionless services. HTTP and ICMP [5] tests have been devised and implemented to measure performance statistically on an ongoing basis. Individuals at organizations such as Intel, Hewlett-Packard and the Pittsburgh Supercomputing Center have developed software to perform these tests and ISPs have been asked to cooperate in these efforts. This memo addresses the need for a basic, repeatable, end-user test capability requiring minimal external support.

(2) A great many users access the Internet via analog dial-up links. To achieve acceptable performance, these connections depend to a large extent on link data compression algorithms implemented in modems. Again, experience has shown that there are not only variations between these algorithms, but also in their implementation and execution by modems from different vendors. The smallest modem configuration errors can result in a loss of data compression, incorrect serial port speed settings, which cannot take full advantage of the compression algorithms, etc.

(3) Various script files have been developed and packaged with remote access application software. These scripts are intended to optimally configure each vendor’s modems under the control of the applications. Often times however, due to the variations noted above, as well as the large number of modem types currently in use, the applications’ scripts do *not* configure the modems optimally. Status messages generated by modems are also
configurable and inconsistent. Often times they are not displayed correctly via the applications’ user interfaces. This causes inaccurate information about the status of the dial-up connection to be reported to the user. Such errors can lead a user to believe that he is achieving optimal performance when, in fact he is not, or that he is not achieving optimal performance when, in fact he is.

Finally, service providers may not support the highest available serial port speeds (or their equivalent) on their side of the dial-up connection. For example, a connection of "28.8 kbps" should be capable of carrying compressible data at two to four times that rate with current modem compression algorithms. This can only occur if user hosts, modems and service provider equipment (i.e. modems, remote access servers, etc.) are configured to work at the highest available serial data rates - *not* the analog wire speeds of the modems. To achieve and verify the maximum possible throughput, the test data samples in the HTML documents described herein were designed to be highly compressible. (Modem compression can always be disabled by end users if desired.)

2. Discussion

2a. Advantages

This memo suggests a methodology using HTML, JavaScript Ver. 1.1 [3], *any* HTTP server and the Netscape Navigator Ver. 3.01 or 4.01 browser to perform end-to-end one-way data throughput and two-way response time tests at the application layer. This software is "off the shelf". It is anticipated that later versions of this user agent will continue to support these tests with little or no changes to the measurement application. No other software or hardware, save that normally resident on HTTP clients and web servers, is required, nor is access to the server’s cgi-bin directory.

Using the methodologies described herein, Test Data Samples are contained in standard HTML files (web pages). Measurement Indicators (timestamps) and the Measurement Application itself are contained in the same web pages as the Test Data Samples. These are written in JavaScript Ver. 1.1, a language specifically designed to be integrated with, and embedded in, HTML.

Unlike some other HTTP GET tests, those documented herein rely on HTML test files of predetermined size and composition. This gives the tests a high degree of repeatability across various Internet clients and servers. The use of standardized web documents also ensures that the throughput test data sample is compressible and that changes to the sample data can, at the very least, be controlled.
To minimize the size of the file used to test User Response Time, JavaScript functions are pre-compiled in a separate test launch file. The resulting test data sample is only 80 Bytes—small enough to be carried in a single TCP packet with a default 536-Byte MSS [4], including the HTTP response header.

With respect to the throughput test, a test data sample size of 96 kB would result in target (ideal) transit times of 80 seconds at 9.6 kbps and 500 milliseconds at T1 (1536 kbps), making this sample size useful to the vast majority of Internet users.

It is possible to load the HTML files on *any* web server and generate measurement data on *any* compliant web browser. Both HTML and JavaScript are not platform or OS dependent and versions of the required user agent have been developed for a wide variety of client systems.

In order to allow end users to obtain empirical real-time measurements from *their* perspective, testing is performed on actual HTTP clients, rather than lower level network entities (e.g. routers, links and clouds), or other hosts. When viewed from the lower level perspectives, these measurements can be said to be *relative* or *derived*. However, from the *end user perspective*, since the test data samples, measurement indicators and measurement applications are themselves comprised of typical user data (HTML and JavaScript), these measurements can be said to be *absolute*. When the measurement perspective is that of the end user, weaknesses in user agents, servers, production TCP, HTTP, etc., which would contribute to measurement errors at lower levels, are not significant as they too are being measured.

The only clock used is that of the client so there are no clock synchronization requirements as there are with one-way delay tests.

A pseudo-random Poisson sampling technique is employed to request repetitive test samples at unpredictable intervals over user-defined time periods. This makes it difficult for an observer to guess the sample request times and makes synchronization with other network events, which may affect measurement quality over time, unlikely.

2b. Caveats

Given that the client computer’s user agent initiates and timestamps its requests, and also timestamps, interprets, calculates and displays the delays and flow rates of the test data from the server, these tests can be said to have absolute accuracy only from the end user’s perspective. When compared to measurements of lower level events (e.g. packet arrival or "wire" times) by external devices (e.g. packet filtering protocol analyzers), differences may be
observed. When run repeatedly with a given client/server pair however, these tests provide realistic performance benchmarks that should not change over time, other things being equal.

In cases of unacceptable or deteriorating performance, testing can continue using different clients and/or servers, other layers of the IP suite and/or other tools (e.g. protocol analyzers, ICMP messages [5], reference congestion control algorithms [6], etc.) to determine the exact cause of the under-performance.

As with any time-sensitive application, for best results, no other tasks should be run concurrently on the client during testing (including the screen saver). Testing requires only a client’s browser and IP stack to be active.

Collection of a statistically significant number of samples requires repeated transfers of the test files over a given time interval from the desired server. If the test files are cached anywhere along the path between the client and the server, results will not be equivalent to those obtained using the full path to the original source server. Caching by the user agent is nullified by discarding results from the initial sample and then using a Reload method, but intermediate caching servers are not under the control of the client.

Caching servers may be used by ISPs to legitimately improve network performance (e.g. between continents), but others (e.g. servers located on user premises) will interfere with the operation of these tests as the source data will often times *not* be retrieved via the Internet. HTTP Version 1.1 [7] allows greater caching control, including the ability to designate files as non-cacheable. These enhancements to the protocol may motivate the development of future versions of these tests.

2c. Statistical Sampling

The following is a discussion of the sampling techniques used by the author for data collection. The techniques for analyzing the collected data are left to the user. Section 4 will show how the test results can be transferred to a spreadsheet or other client-resident analysis tool.

A random sampling technique is used to ensure that results are as unbiased as possible. JavaScript has the capability of generating pseudo-random numbers, which are used to set random inter-sample intervals in both tests described herein, immediately after each sample is received. A discussion of the criteria used for selection of the average size of the inter-sample intervals follows:

The User Response Time test data file is small (80 Bytes), so
frequent repetitive GETS of the file will not impact network loading substantially. Since the User Data Throughput test contains more bytes per sample (at least 96 kB), randomized samples with longer average inter-sample intervals are employed. To keep to the objective of a real-time test that can be meaningful to the "end user", an optional singleton method is made available to GET the latter file with a user-initiated manual override prior to the expiration of any inter-sample interval.

The value of 96 kB was chosen to simulate a large HTML document. This is suitable for measuring throughput from 9.6 kbps to T1 (1536 kbps), or more, with reasonable accuracy on almost any client. Larger test data samples can be used to measure higher throughput rates if desired.

Only one TCP connection is used per sample. This parallels the stated direction of future HTTP implementation [7] and will therefore become an even better representation of "real world" web pages over time. Tests using multiple embedded objects could also be developed.

In both tests, Poisson sampling is approximated by repeatedly generating GET requests for each test data sample at random intervals, averaging a fixed number of requests over a given time duration. The random intervals and average frequency of the requests are set by the Measurement Application, while the total time duration of the test is determined by the user. Auto timeout options of 1/2 Hour, 1 Hour, 2 Hours, 4 Hours, 8 Hours and 1 Week are provided.

Although the Poisson intervals are set, they may vary as a result of the longer transfer times which may occur during busy periods. This could result in fewer samples taken during periods of slow performance, thereby skewing the averaged results. One possible solution to this problem would be to set and start the subsequent inter-sample interval *before* the pending requested sample is received, but this could result in several samples being received at once, (also skewing results), or, as is the case with web browsers, an interrupted transfer of the pending sample. Neither of these conditions would be desirable during the testing process.

A better alternative would be to set the average inter-sample interval to be much larger (e.g. an order of magnitude) than the expected average response time (or expected total transit time in the case of the throughput test) at the application layer. For example, a Response Time test with an average interval of 18 seconds would yield about 200 samples per hour. With an expected 1.8-second average result, this would actually be implemented by setting the average interval to 16.2 seconds immediately after a sample is received. The author has chosen this setting for Response Time tests, where the order of magnitude rule should suffice. Of course, it is not possible
to know a priori what the average response time or throughput will be so any setting of the Poisson interval would be an educated guess.

This is more complicated in the case of the Throughput Test, because available bandwidth plays such a major role in affecting the result. Bandwidth can vary widely (i.e. by several orders of magnitude) by physical connection type, congestion level, etc. Since the Throughput Test file is many times larger than the Response Time Test file, a longer interval (less sampling) is desirable so as not to influence end-to-end loading, but in no case should fewer than 20 samples per hour be taken. This makes inter-sample intervals that are very long with respect to transfer times impractical at slower speeds. The above notwithstanding, the prudent course would seem to be to make the average inter-sample interval at least somewhat longer than the file transfer time of the slowest expected connection (i.e. analog dial-up, poor line quality, sans data compression - about 9.6 kbps).

Given the above, for the 96 kB Throughput Test, the author has chosen an average inter-sample interval of 120 seconds. Variations in bandwidth could allow an average of only 18 samples per hour to be taken at 9.6 kbps, assuming zero round trip delay, and a maximum average of 30 samples per hour in the hypothetical case of a network with infinite bandwidth and zero delay. Adding fixed delay values to these assumptions and changing the maximum throughput to a value less than infinity (e.g. T1), reduces the variations in sampling rates at various throughput values, but they are still quite significant.

The implementation of the combined Response Time/Throughput Test described herein uses the following Adaptive Poisson Sampling technique to address this problem:

Since the client shall not send a request for the next sample until the current pending one is received, slow connections will allow fewer samples than fast connections, tainting the Poisson algorithm. By adjusting the average random inter-sample interval dynamically, after the receipt of each sample, depending on the time the sample took to arrive, a more constant random sampling rate can be maintained. For example, if a file took 30 seconds to be transferred to the client, an average inter-sample interval of 120 seconds (30 per hour) could be shortened to 90 seconds so that over time, the 30 per hour sample rate will be maintained. Since measurement of the file’s transfer time is implicit in this test, the adjustment factor is computed and applied after each sample is received. Response time is compensated for in the same manner.

Other work has been undertaken to define methods to statistically compensate for the reduction in the number of samples received during periods of slow performance, so as not to understate such performance in the analysis phase. The median value and inter-quartile (25th to
75th percentile) range have proven to be useful in this area. For simplicity however, the tests herein produce mean summary results.

The author has defined a 30 second response time-out interval for both tests, beginning at the time of the request for the initial sample. The choice of this value reflects typical user attempts to retry requests after a period of perceived idle time has elapsed. If this timer expires, an error message is displayed by the client and a retry occurs. Error statistics can then be generated based on the number of unsuccessful attempts.

3. Metrics

These metrics are application layer, file copy metrics and should not be confused with others developed for use at the network and/or transport layers. It is assumed that all requests are made to the same Domain Name (host) and that name resolution has been completed.

3a. User Data Throughput

At Time0, a file containing a throughput test data sample of N kBytes is requested from the server by the client. At Time1, the first byte of the file’s throughput test data sample is received by the client from the server. At Time2, the last byte of the sample’s contents is received by the client from the server. dTime3 is defined as the difference, in seconds, between Time2 and Time1, as measured by the client. The User Data Throughput in kbps is defined as 8*N/dTime3.

3b. User Response Time

At Time0, a file of N Bytes where N<(MSS-HTTP header) contained in a single TCP packet is requested from the server by the client. At Time1, the file is received by the client from the server. dTime2, the User Response Time, is defined as the difference between Time1 and Time0 in milliseconds, as measured by the client.

3c. Combined User Response Time/Data Throughput

Both of the above metrics can be combined as follows to allow measurement of correlation between them:

At Time0, a file containing a throughput test data sample of N kBytes is requested from the server by the client. At Time1, the first byte of the file’s contents is received by the client from the server. dTime2, the User Response Time, is defined as the difference between Time1 and Time0 in milliseconds, as measured by the client.
At Time3, the first byte of the file’s throughput test data sample is received by the client from the server. At Time4, the last byte of the sample’s contents is received by the client from the server. dTime5 is defined as the difference, in seconds, between Time4 and Time3, as measured by the client. The User Data Throughput in kbps is defined as 8*N/dTime5.

3d. Some Other Interesting Derived Metrics

Knowing how the total transaction time is divided between initial response time and subsequent data transfer time, is useful in determining likely causes of performance anomalies, especially if costly alternatives are being considered to improve performance. For example, if the major portion of the total transaction time is due to response time rather than transfer time, adding more bandwidth to the network will probably not improve performance significantly.

Given the metrics in Section 3c, it is a simple matter to derive:

- dTime6, the Total Transaction Time, in seconds, defined as, dTime2+dTime5.

We can then express dTime2 and dTime5 as percentages of the Total Transaction Time:

- 100*dTime2/dTime6 is defined as the percentage of User Response Time.
- 100*dTime5/dTime6 is defined as the percentage of User Data Throughput Time.

4. Implementations of Test Methodologies

The author’s implementation of the tests consists of three web server files and a Results Window which is generated on the client side in JavaScript. Timestamps are inserted to conform to the metrics in Section 3 as closely as possible. All or part of the contents of the Results Window can be saved as a text file or spreadsheet for subsequent analysis. A menu bar with a Copy option is provided for this purpose as part of the Results Window GUI. To observe possible correlation between response time and throughput measurements, the combination test described in Section 3c is implemented as are the metrics derived in Section 3d. The ability to measure both parameters of each sample aids in determining likely causes of performance anomalies.

The following summarizes the author’s implementations of the tests. They are more fully documented in Section 9. Suggested file names appear below and in Section 5.
4a. Test Launch Page (ihttt.htm)

This initial web page contains a description of the tests in HTML and the JavaScript functions which open the client side Results Window. Button objects are defined to trigger onClick Event Handlers which call the functions. The user is offered a choice of a Response Time only test or a combination Response Time/Throughput test. When called, these functions in turn, write HTML and more functions, to the Results Window. The latter functions persist, even after their parent window’s URL has been changed to load the test data sample pages. The persistent functions perform as follows:

For both tests:

1) offer the user a choice of test durations and set the selected termination time;
2) initialize sample counter and results adders to zero (used to calculate mean summary results);
3) request test data sample from the server;
4) get the time of each request (per the client’s internal clock);
5) get the file’s arrival time (at the application layer, per the client’s internal clock);
6) calculate the time difference between the last pending request and the file’s arrival (dTime2);
7) display the sample’s arrival date and time in HTML in the Results Window;
8) display the Response Time test result (dTime2) in milliseconds in HTML in the Results Window;

For the Response Time Test:

9) ignore the first sample received if its transfer time was <1 second, (possibly locally cached); if not, use it;
10) calculate the next Poisson inter-sample interval;

For the combined Response Time/Throughput Test:

11) get Time3 (section 3c) per the client’s internal clock);
12) receive the throughput test data sample;
13) get Time4 (section 3c) per the client’s internal clock);
14) calculate dTime5 and dTime6 (section 3c);
15) ignore the first sample received if its transfer time (dTime6) was <10 seconds, (possibly locally cached); if not, use it;
16) calculate User Data Throughput in kbps and display it in HTML in the Results Window;
17) calculate the percentage of Total Transaction Time for Response Time and Transfer Time and display them in HTML in the Results Window;
18) calculate the next Adaptive Poisson inter-sample interval;
For both tests:

19) request the next sample (reload document from server);
20) get the time of the request (per the client’s internal clock)
21) if the next sample is not received in 30 seconds, display an error message in HTML in the Results Window and reload the document from the server by repeating item 3 above;
22) upon test completion, compute and display mean summary results.

The HTML/JavaScript code for this page, with comments, appears in Section 9b (1).

4b. User Response Time Page (delay.htm)

All necessary functions are called from the Results Window when this document is loaded. A Button object is defined and written to this page on the client side to allow the user to terminate the test at will. Calling persistent, pre-compiled functions from the Results Window allows the size of this file to be limited to 80 Bytes (one line of text). The file can be contained in a single TCP packet, including all headers.

The HTML/JavaScript code for this page, with comments, appears in Section 9b (2).

4c. Combined User Data Throughput/Response Time Page (thrpt.htm)

This page triggers both response time and throughput measurements. Button objects are defined to allow the user to terminate the test at will, or to request the next sample prior to the expiration of any Adaptive Poisson interval. All the functions called by this page are pre-compiled in the Results Window. This page contains 96kB of compressible test data and some descriptive HTML.

The HTML/JavaScript code for this page, with comments, appears in Section 9b (3).

5. Test File Names

In order to make these, or other similar test files, easily accessible to Internet users wishing to run the tests from a given server, the following is suggested as a file naming convention. For the example host www.isi.edu:

Test Launch Page: www.isi.edu/rfcNNNN/ihttt.htm
User Response Time Page: www.isi.edu/rfcNNNN/delay.htm
Combined User Data Throughput/Response Time Page: www.isi.edu/rfcNNNN/thrpt.htm
6. Security Considerations

This memo raises no security issues.

7. References


[3] Netscape JavaScript Reference
http://home.netscape.com/eng/mozilla/3.0/handbook/javascript/


8. Author's Address

Howard Stanislevic
HSCOMMS Network Engineering
and Consulting
15-38 146 Street
Whitestone, NY 11357

Phone: 718-746-0150
EMail: hscomms@aol.com

9. Appendix

9a. Sample Test Results Output (Combined Test):
+----------------------------------------------------------------------+
<table>
<thead>
<tr>
<th>Netscape - {Test Results}</th>
</tr>
</thead>
<tbody>
<tr>
<td>File  Edit  View  Go  Bookmarks  Options  Directory  Window  Help</td>
</tr>
<tr>
<td>----------------------------------------------------------------------</td>
</tr>
<tr>
<td>Thank you. You have selected a 1 Hour test.</td>
</tr>
<tr>
<td>----------------------------------------------------------------------</td>
</tr>
<tr>
<td>Response Time and Throughput for a 96kB File</td>
</tr>
<tr>
<td>Retrieved at Random Intervals Averaging 120 Seconds</td>
</tr>
<tr>
<td>----------------------------------------------------------------------</td>
</tr>
<tr>
<td>Date</td>
</tr>
<tr>
<td>---------</td>
</tr>
<tr>
<td>7/31/97</td>
</tr>
<tr>
<td>7/31/97</td>
</tr>
<tr>
<td>7/31/97</td>
</tr>
</tbody>
</table>

Total Samples: 3
Average Response Time: 838 msec
Average Throughput: 840 Kbps

You have ended the test. To save all or part of this data, select Edit from the Menu bar above.

9b. HTML and JavaScript code - Interactive Hypertext Transfer Tests

Variables used in the following scripts are defined in Section 3. Operation of the tests is summarized in Section 4. New lines and indented text are used for clarity and should be deleted before dynamically writing to a window.

(1) Test Launch Page (ihttt.htm):

```html
<html>
  <head>
    <title>Interactive Hypertext Transfer Tests</title>
    <script>
      /* Offer user choice of test durations, flag first sample, declare functions for no response timeout, test reload, test end by user, test end by timer. */
      function winOpen () {
        o.window.focus();
      }
    </script>
  </head>
  <body>
    <!-- Body content goes here. -->
  </body>
</html>
```

Stanislevic
<B>For how long would you like to run the test?<FORM>
&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&lt;/OPTION&gt;
&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nmber document.write('</OPTION></FORM>
&</B></CENTER></FONT></P>

o.document.write("<P><CENTER><FONT COLOR="#FFFF00">&lt;B>For how long would you like to run the test?<FORM>
&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&n...
and Throughput*/
Flag=3D1;
Count=3D0;
totalRT=3D0;
totalTh=3D0;

/*Display possibly cached warning and reload document.*/
function noCache() {
    document.write('<HR WIDTH=3D100%><B>First sample may have been locally =

    cached. Reloading from server...</B><BR>);
    window.scroll(0,1080);
    testReload();
}

/*If no response received from server, display error message and reload
document.*/
function noR() {
    document.write('<B>No response from server for 30 seconds. =

    Retrying...</B><BR>);
    window.scroll(0,1080);
    testReload();
}

/*Clear first sample flag, reload document, get new Time0 and set No
Response Timer.*/
function testReload() {
    Flag=3D0;
    window.focus();
opener.location.reload();
    Time0=3Dnew Date();
    Rno=3DsetTimeout('noR()',30000);
}

/*Terminate test when called by the user and return to Test Launch page.*/
function uE() {
    clearTimeout(Rno);
    clearTimeout(Rt);
    clearTimeout(endTest);
    window.focus();
    Summary();
opener.location.replace('ihttt.htm');
document.write('<CENTER><FONT COLOR=3DFFFF00>
<T>B> You have ended the test. To save all or part of this data,<BR>
select Edit from the Menu bar above.</B></FONT></CENTER>
<HR WIDTH=3D100%'>);
    window.scroll(0,1080);
}

/*Terminate test when time expires and return to Test Launch page.*/
function testEnd() {
    clearTimeout(Rno);
    clearTimeout(Rt);
    Summary();
}
opener.location.replace('ihttt.htm');
document.write('<CENTER><FONT COLOR=3DFFFF00><B>Test Complete! =
To save all or part of this data,<BR>
select Edit from the Menu bar above.</B></FONT></CENTER>
<HR WIDTH=3D100%>');
window.scroll(0,1080);
}
</SCRIPT>

/*Open Results Window, declare Response Time Test functions. Create user
Stop Test button in main window.*/
function testDelay() {
  o=3Dwindow.open("","Rsl","height=3D250,width=3D640,scrollbars=3D1,status=
3D0,
toolbar=3D0,directories=3D0,menubar=3D1,resizable=3D0");
o=document.close();
o=document.write("<HEAD><TITLE>Test Results</TITLE></HEAD>
<BODY TEXT=3D'#FF0000' BGCOLOR=3D'#000000'>
<SCRIPT>
/*Get arrival time, calculate dTime2. If this is a first sample AND
transfer time was short, discard data, otherwise display date/time and
Response Time, create user Stop Test button, increment sample counter,
compute new Total Response Time, set Poisson inter-sample interval
between 0 and 32.4 seconds.*/
function A() {
  Time1=3Dnew Date();
clearTimeout(Rno);
dTime2=3D(Time1-Time0);
If (Flag=3D=3D1 && dTime2<1000) {
  noCache()
} 
else {
  B()
}
function B() {
  dat1=3DTime1.getDate();
  mon1=3D1+Time1.getMonth();
  yea1=3DTime1.getYear();
  hou1=3DTime1.getHours();
  min1=3DTime1.getMinutes();
  sec1=3DTime1.getSeconds();
document.write(’<CENTER><TABLE BORDER=3D1 CELLPACING=3D0 WIDTH=3D65%>
<TR ALIGN=3DCENTER>
<TD ALIGN=3DCENTER WIDTH=3D13%’,(mon1),’/’,(dat1),’/’,(yea1),’</TD>
<TD ALIGN=3DCENTER WIDTH=3D20%’,(hou1),’:',(min1),’:',(sec1),’</TD>
<TD ALIGN=3DCENTER WIDTH=3D32%’,(dTime2),’</TD>
</TR></TABLE></CENTER> ‘);
window.scroll(0,1080);
opener.document.write ('<br><br><br><br><br><br><br><br><br><br><br>
<center><form><input type=3Dbutton =
value=3DSTOP &nbsp;and &nbsp;Return &nbsp;to &nbsp;Test &nbsp;Menu =
onClick=3Do.uE()></form></center>);
Count++;
totalRT+=3DdTime2;
Rt=setTimeout('testReload()',(Math.random()*32400));
}
/*Start test, get Time0, set No Response timeout*/
function testStart() {
    document.write('<HR WIDTH=3D100%><CENTER><FONT COLOR=3D3C9CE3>
    <B>Response Time for an 80-Byte File<BR>
    Retrieved at Random Intervals Averaging 18 Seconds</B></FONT><HR WIDTH=3D100%><TABLE CELLSPACING=3D0 WIDTH=3D65%><TR>
    <TD ALIGN=3DCENTER WIDTH=3D13%><FONT COLOR=3DFFFF00><B>Date</B></FONT>=
    </TD>
    <TD ALIGN=3DCENTER WIDTH=3D20%><FONT COLOR=3DFFFF00><B>Time</B></FONT>=
    </TD>
    <TD ALIGN=3DCENTER WIDTH=3D32%><FONT COLOR=3DFFFF00>
    <B>Response Time (msec)</B></FONT></TD></TR></TABLE>);
    window.scroll(0,1080);
opener.window.location.href=3D('delay.htm');
    Time0=new Date();
    Rno=setTimeout('noR()',30000);
}
/*Compute and write Average Response Time.*
function Summary(){
    avRT=Math.round(totalRT/Count);
    document.write('Total Samples: ',Count,'
Average Response Time: ',avRT,' msec
</HR WIDTH=3D100%>
</B>');
    winOpen();
}
/*Open Results Window, declare combined Response Time/Throughput Test
functions.*/
function testThrpt() {
    o=3Dwindow.open('',"Rsl","height=3D250,width=3D640,scrollbars=3D1,status=
    3D0,
toolbar=3D0,directories=3D0,menubar=3D1,resizable=3D0");
o.document.close();
o=document.write("<HEAD><TITLE>Test Results</TITLE></HEAD>
<BODY TEXT=3D#FF0000 BGCOLOR=3D'#000000'><SCRIPT>
/*Calculate dTime5, dTime2 and dTime6. if this is a first sample AND
transfer time was short, discard data.*/
function ifFirst() {
    dTime5=3D(Time4-Time3);
    dTime2=3D(Time1-Time0);
    dTime6=3D(dTime2+dTime5);
If (Flag=3D=3D1 && dTime6<10000) {
    noCache()
} else {
    APS()
}

/*Reload test when called by user.*/
function userReload() {
    clearTimeout(Rt);
    testReload()
}

/*Get arrival time*/
function aT() {
    Time1=3Dnew Date();
    clearTimeout(Rno);
}

/*Get Time3*/
function T3() {
    Time3=3Dnew Date();
}

/*Get Time4*/
function T4() {
    Time4=3Dnew Date();
}

/*Display date/time, Response Time, Throughput and % of Total Time, increment sample counter, compute new Total Response Time and Throughput set Adaptive Poisson interval between 0 and (240-(2*dTime6)) secs.*/
function APS() {
    dat1=3DTime1.getDate();
    mon1=3D1+Time1.getMonth();
    yea1=3DTime1.getYear();
    hou1=3DTime1.getHours();
    min1=3DTime1.getMinutes();
    sec1=3DTime1.getSeconds();
    Thrpt=3DMath.round(8000000/dTime5)/10;
    document.write('<TABLE BORDER=3D1 CELLSPACING=3D0 WIDTH=3D100%>
  <TR ALIGN=3DCENTER>
    <TD ALIGN=3DCENTER WIDTH=3D13%>','(mon1),/(dat1),/(yea1),'</TD>
    <TD ALIGN=3DCENTER WIDTH=3D13%>','(hou1),:(min1),:(sec1),'</TD>
    <TD ALIGN=3DCENTER WIDTH=3D20%>','(dTime2),'</TD>
    <TD ALIGN=3DCENTER WIDTH=3D21%>','(Thrpt),'</TD>
  </TR>
  <TR ALIGN=3DCENTER WIDTH=3D21%>','Math.round(1000*dTime2/dTime6)/10,'</TD>
  <TD ALIGN=3DCENTER WIDTH=3D21%>','Math.round(1000*dTime5/dTime6)/10,'</TD>
</TABLE>');</p>
window.scroll(0,1080);
Count++;
totalRT+=3DdTime2;
totalTh+=3DThrpt;

Stanislevic
function testStart()
{
  document.write('<HR WIDTH=100%><CENTER><FONT COLOR=3C9CE3>' +
  '<B>Response Time and Throughput for a 96kB File</B>' +
  '<HR WIDTH=100%><TABLE CELLSPACING=3 WIDTH=100%><TR>' +
  '<TD ALIGN=DCENTER COLSPAN=3 WIDTH=26%></TD>' +
  '<TD ALIGN=DCENTER WIDTH=20%><FONT COLOR=FFFF00>' +
  '<B>Response</B></FONT></TD>' +
  '<TD ALIGN=DCENTER WIDTH=12%><FONT COLOR=FFFF00>' +
  '<B>Thrpt</B></FONT>=</TD>' +
  '<TD ALIGN=DCENTER COLSPAN=3 WIDTH=42%><FONT COLOR=FFFF00>' +
  '<B>% of Total Time</B></FONT></TD>' +
  '</TR><TR><TD ALIGN=DCENTER WIDTH=13%><FONT COLOR=FFFF00>' +
  '<B>Date</B></FONT></TD>' +
  '<TD ALIGN=DCENTER WIDTH=13%><FONT COLOR=FFFF00>' +
  '<B>Time</B></FONT>=</TD>' +
  '<TD ALIGN=DCENTER WIDTH=20%><FONT COLOR=FFFF00>' +
  '<B>Time (msec)</B></FONT></TD>' +
  '<TD ALIGN=DCENTER WIDTH=12%><FONT COLOR=FFFF00>' +
  '<B>(kbps)</B></FONT></TD>' +
  '<TD ALIGN=DCENTER WIDTH=21%><FONT COLOR=FFFF00>' +
  '<B>Resp Time</B></FONT></TD>' +
  '<TD ALIGN=DCENTER WIDTH=21%><FONT COLOR=FFFF00>' +
  '<B>Transfer Time</B></FONT></TD>' +
  '</TR></TABLE>');</n
  window.scroll(0,1080);
  opener.window.location.href='thrpt.htm';
  Time0=new Date();
  Rno=setTimeout('noR()',30000);
}

function Summary()
{
  avRT=Math.round(totalRT/Count);
  avTT=Math.round(totalTh/Count);
  document.write('<BR>' +
  '<B>Total Samples: </B>' +
  '<FONT COLOR=FF0000> ',(Count),'</FONT><BR>' +
  '<B>Average Response Time: </B>' +
  '<FONT COLOR=FF0000>',(avRT),' msec</FONT><BR>' +
  '<B>Average Throughput: </B>' +
  '<FONT COLOR=FF0000>',(avTT),' kbps</FONT></B><HR WIDTH=100%>');
}

winOpen();
//--></SCRIPT>
</HEAD>

<BODY TEXT=3D"#FFF000" BGCOLOR=3D"#000000" LINK=3D"#00FFFF" VLINK=3D"#00FFFF" ALINK=3D"#FF8000"}
<CENTER><P><FONT COLOR=#3C9CE3><FONT SIZE=+2><B>Interactive =
Hypertext Transfer Tests</B></FONT></FONT></CENTER>

<P><FONT SIZE=+1>Cool HTTP User Response Time and Throughput Tests you =
can run with your browser!</FONT></P>

<SCRIPT><!--/*Create Button objects and onClick Event Handlers.*-
document.write("<CENTER><P><FONT COLOR=#3C9CE3'><B><FONT SIZE=+2>
How to Run the Tests with JavaScript</B></FONT></FONT>
<HR WIDTH=50%'></P></CENTER>

<P><FONT SIZE=+1>Navigator 3/4 JavaScript Users: Just select the =
test(s) you would like to run.</P>

<FORM><INPUT type=button name=button1 value=Test Response Time Only' onClick=testDelay()>
<INPUT type=button name=button2 value=Test Response Time and =
Throughput' onClick=testThrpt()></FORM></P></CENTER>
--></SCRIPT>

(2) User Response Time Page (delay.htm)

<html>
<script>
o=open("","Rsl","scrollbars=3D1,menubar=3D1");
/*Call function A from the Results Window.*/
o.A();
</script>
</html>

(3) Combined User Data Throughput/Response Time Page (thrpt.htm)

<html>
<head>
<script>
o=open("","Rsl","scrollbars=3D1,menubar=3D1");
/*Call function aT from the Results Window.*/
o.aT();
</script>
</head>
<body text=3D"#fff000" bgcolor=3D"#000000" link=3D"#00ffff" =
vlink=3D"#00ff00" alink=3D"#ff0000">
<script>
/*Call function T3 from the Results Window.*/
o.T3();
</script>

<!--96,000-Byte compressible test data sample (96,000 dots in an HTML =
comment tag, not displayed).................................-->

<script>
/*Call function T4 from the Results Window.*/
o.T4();
</script>

<center><p><b><br><br><br><br><font color=#ff0000>
<font size=+4>DONE!</font></font></b><br></p></center>

<p><b>The Test will now restart automatically at random intervals =
averaging 120 seconds. If you prefer instead to STOP the test or =
run the test again NOW, click one of the buttons below: </b></p>

<!--Create Button objects and onClick Event Handlers.-->
<center>
<form><input type='button' name='button1' =
value='STOP and Return to Test Menu' onClick='o.uE()'</form>&nbsp;
<form><input type='button' name='button2' value='Run Test Again NOW=' =
onClick='o.userReload()'</form>
</center>

<script>
/*Call function ifFirst from the Results Window.*/
o.ifFirst();
</script>
</body>
</html>