firehunter White Paper

Measuring Web Quality of Service with the Firehunter HTTP Test

This white paper describes the Firehunter HTTP test and its use in measuring Web Quality of Service (QoS). In addition, this white paper explains the similarities and differences between the HTTP test and common Web browsers. (Note that the HTTP test also supports HTTPS.)

This information should be helpful to any person deploying or using Firehunter, including Network/System Administrators, as well as business managers and customers of service providers. In particular, this information may help you with troubleshooting, reporting, configuring service level agreements (SLAs), and planning. It is assumed that you have a basic understanding of Firehunter and of HTTP, HTML, and other Internet technologies.

This white paper contains the following main topics:

Agilent Technologies

Introduction

Firehunter's HTTP test is an important and widely used test due to the pervasiveness of HTTP-based Internet technology. The HTTP test downloads a Web page from a server in a manner similar to a Web browser. However, the HTTP test differs from a browser primarily because the test then takes measurements that are indicators of the QoS provided by the HTTP service.

To be specific, the goal of the Firehunter HTTP test is to provide a consistent, accurate set of measurements that help gauge the availability and performance of a Web server over time. The HTTP test also provides information to help isolate the root cause of any problem that occurs. The test does this in a repeatable, scalable manner that is critical for internal or external service providers that must conform to service level agreements (SLAs). The default behavior of the test is to direct proxies to send the request on to the Web server and not return a cached copy of the requested object. Also the test does not send directives that result in "Not Modified" responses, so that the object requested is downloaded each time the test is run. When combined with Firehunter baselines and thresholds, the measurements provided by the test can accurately detect changes in the service. Relative differences from the baseline indicate real Web QoS changes that may indicate problems.

In contrast, the goal of a browser is simply to display a Web page as fast as possible while being lenient in terms of HTTP or HTML errors. To achieve this goal, browsers, along with proxies, utilize various caching algorithms for Web requests. This results in better performance in rendering Web pages with a browser. A side effect of these strategies is that the exact transaction that takes place is not the same each time a Web page is downloaded. For example, a browser may send a request to a server that includes cache control directives. The server may send a "Not Modified" 1 response instead of actually returning the requested object. This will have a direct bearing on the time it takes the transaction to complete. Intermediate proxies may also have a cached copy of the object and the request may never get to the actual server. Thus, transaction times seen by users may vary widely based on their unique browser, network, and proxy configurations, as well as their recent browsing history. Because of this type of variability, characterizing abnormal service behavior is clearly more difficult via a Web browser.

The following sections explain the methodology used to obtain measurements with the Firehunter HTTP test, the configuration options for the test, and the recommended use and benefits of the test. This information will help you better understand the meaning and significance of the measurements that the HTTP test provides.

Test Methodology

The HTTP test actively samples a Web service at a given frequency using HTTP 1.0 protocol and provides:

- Measurements that attempt to estimate QoS and quantify the important components of the test activity
- Baselines (i.e. running averages) and thresholds to identify normal behavior by time-of-day and day-of-week
- An extensive set of configuration options that allow you to test various aspects of the HTTP activity

The HTTP test issues an HTTP GET request to the chosen Web server for the desired Web page. It then reads the HTTP response from the server. If the response to the GET is redirected, then the test follows that redirection, sending a GET for the new page.

The test maintains and manages its own list of HTTP Cookies that are sent as part of the HTTP requests. The test implements the Cookie list according to the HTTP Cookie specification RFC 2965 2. The format of expiration dates and other aspects of Cookies are one area where Firehunter adheres more strictly to the specification than most Web browsers. This feature has helped users of the test to identify Web site problems quickly and proactively.

While performing these tasks, the test takes a number of measurements that are reported to the Firehunter Diagnostic Measurement Server (DMS). The following measurements are defined in Appendix A.

- Availability
- TotalResponseTime

QoS measurements

- DnsTime
- TcpConnectTime
- AdditionalTcpConnectTime
- ServerResponseTime
- DataTransferTime
- DataTransferRate
- TestCompleted
- InitialDownLoadTime
- ObjectDownLoadTime
- EmbeddedObjectErrors

Diagnostic measurements

The first two measurements, Availability and TotalResponseTime, address the QoS of the Web service. The remaining measurements can be used to help diagnose problems when they occur, as described in *Measurement and Management of Internet Services Using Agilent Firehunter 3*. The Measurements section later in this white paper provides the details of how all the measurements are taken. The test properties that you can configure are described in the next section, Test Configuration.

The HTTP test can be used for both normal (HTTP) and secure (HTTPS) Web pages. HTTP 1.0 is supported; persistent connections as implemented with HTTP 1.1 are not used. Java script is not parsed by the test, but the test adheres more closely to the HTTP and HTML specifications than most Web browsers. This feature can help identify problems with Web sites that may be hidden by certain Web browsers.

Test Configuration

The HTTP test emulates the topology of a Web browser using an HTTP server. For example, it can connect either directly to a Web server or indirectly through a proxy. The test supports a set of configuration options designed to control HTTP interaction with the server. The following list identifies the test options and how they affect the behavior of the test:

- Host Name or IP Address – identifies the Web server
- Port – HTTP port number on server (default is `80`)
- Page – Web page to be tested (default is `/`)
- Proxy – Web proxy and port if needed
- ProxyCache – turn on or off the use of caching at the proxy server (default is `Off`)
- ProxyUser – username if proxy authentication is used
- ProxyPassword – password for proxy authentication
- User – username if Web site authentication is used
- Password – password for Web site authentication
- DownLoadType – download just initial page, or embedded objects also
- UserAgent – HTTP header field (default is `Mozilla/4.0`)
- HostHeader – used to specify raw HTTP header values
- MustHave – strings that must be contained in the downloaded material
- FailsWith – strings that should not appear in the downloaded material
- DownLoadThreads – number of parallel threads to use when downloading embedded objects

Configuration Implications

ProxyCache is one option that can have a significant impact on the results of the HTTP test. If a proxy is used in the test, the caching capabilities of the proxy can have a significant effect on the reported response times. Since the goal of the test is to measure the QoS of the Web service and not the proxy caching, the default option is to direct the proxy to not use its caching.

DownLoadType is another option that can have a large impact on the test. It controls whether the test downloads just the initial Web page, or the page and any embedded images and frames that may be in the initial page. In the latter case, images and frames are downloaded; however items such as style sheets are not included. If embedded objects reside on other Web servers, the performance of those servers will be reflected in the response times reported.

MustHave and FailsWith should be left blank unless it is important to verify the exact contents of the HTML returned. This can be important for Web sites that return a separate error page when the request has errors. In these situations, either the correct page or error page would return a status of 200 (OK). The test must be configured to differentiate based on the content of the HTML. In cases like this, the MustHave string should be set to a string that is known to exist in the HTML of the correct page. Alternately, the FailsWith string could be used to identify a string known to exist in the HTML of the error page.

When configured, the MustHave and FailsWith options are used to control the ValidPage-Content measurement. ValidPageContent is 100 percent if the MustHave string is found in the downloaded content and zero percent if it isn't. ValidPageContent is zero percent if the FailsWith string is found in the downloaded content and 100 percent if not.

The test supports basic authentication if required by a proxy or Web site. The User, Password, ProxyUser, and ProxyPassword options control this functionality.

The DownLoadThreads is an option implemented in Firehunter version 3.3 that controls how many threads are used to download the embedded objects found in the Web page. Using more than one parallel thread more closely approximates the behavior of a Web browser, which typically uses parallel threads to download embedded objects. This can make the response times of the test closer to the time a browser would take to download the Web page. This feature is discussed further in the next section.

Measurements

The HTTP test measurements are structured the same as many other Firehunter tests – that is, it has Availability and TotalResponseTime measuring QoS, as well as other component measurements for diagnostics.

The measurements that the HTTP test takes are the same whether the Web site to be loaded is a secure (HTTPS) site or not. The only difference in the methodology between HTTPS and HTTP is the use of a Secure Sockets Layer (SSL) when connecting to secure sites. The test supports SSL 3.0 and TLS. SSL 2.0 and earlier protocols are not supported.

Each time the test runs, it performs the following set of steps:

1 Perform address lookup for hostname.
2 Open connection (socket) to server or proxy.
3 Send GET request for URL.
4 Read response from server.
5 Parse HTML returned.
6 Download embedded objects (sequentially or concurrently, using multiple threads).

When the DownLoadType option is set to `HTMLOnly`, only steps 1 – 4 are executed. When DownLoadThreads option is set to one, the embedded objects are downloaded sequentially, instead of in parallel. Sequential downloading was the only behavior available in Firehunter 3.2.1 and earlier versions.

When run in the sequential download manner (Figure 1), the test first performs a name lookup of the hostname. DnsTime is the time to execute this lookup. Next, the test opens a socket to the server or proxy. TcpConnectTime is the time required to open the socket.

Once the socket is open, the GET request is written for the desired URL. This GET request includes UserAgent and Host header fields that are configurable. ServerResponseTime is the time it takes to read the first byte back from the GET request. This time does not include the time to send the request, as that is influenced by the client performance. ServerResponse-Time is calculated starting from after the GET is sent, until the first byte of the response is received. In Firehunter 3.2.1 and earlier, ServerResponseTime was calculated in a similar fashion, but from a separate HTTP HEAD request. The GET request more closely represents a normal HTTP transaction.

DataTransferTime is the time it takes to read the entire response from the GET request. It includes the ServerResponseTime, plus the additional time to read the remainder of the initial web page. As the initial page is parsed, a list of embedded images (<img> tag) and frames (<frame> tag) are constructed. These objects are then downloaded from the appropriate server, which may be different from the initial Web server. If duplicate objects are in the list, the test only downloads the object once. For each object downloaded, the time to open the socket is added to the AdditionalTcpConnectTime and the time to download the object is added to the DataTransferTime.

TotalResponseTime is calculated by summing the DnsTime, TcpConnectTime, Additional-TcpConnectTime, and DataTransferTime. ServerResponseTime is not explicitly added into TotalResponseTime because it is a component of DataTransferTime. TotalResponseTime does not equal the execution time of the test because the measurements are designed with the QoS and diagnostic capabilities in mind. The performance of the client system, where the test is running, affects the execution time, but it is not a measure of the server QoS.

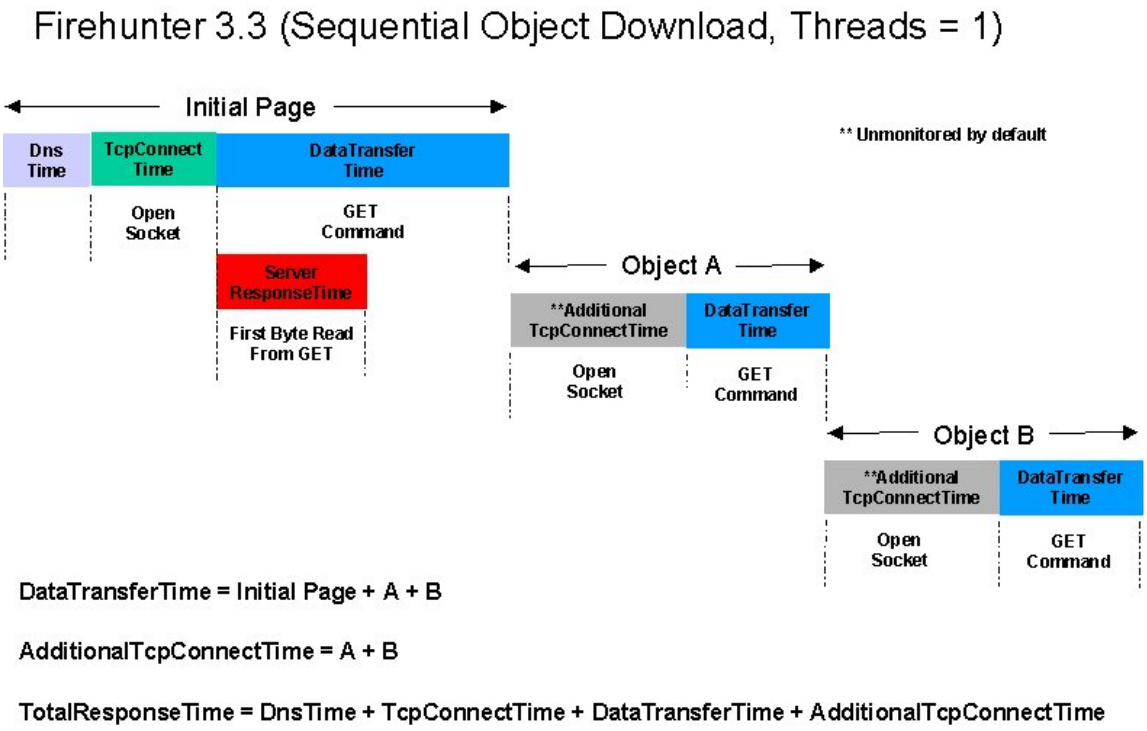
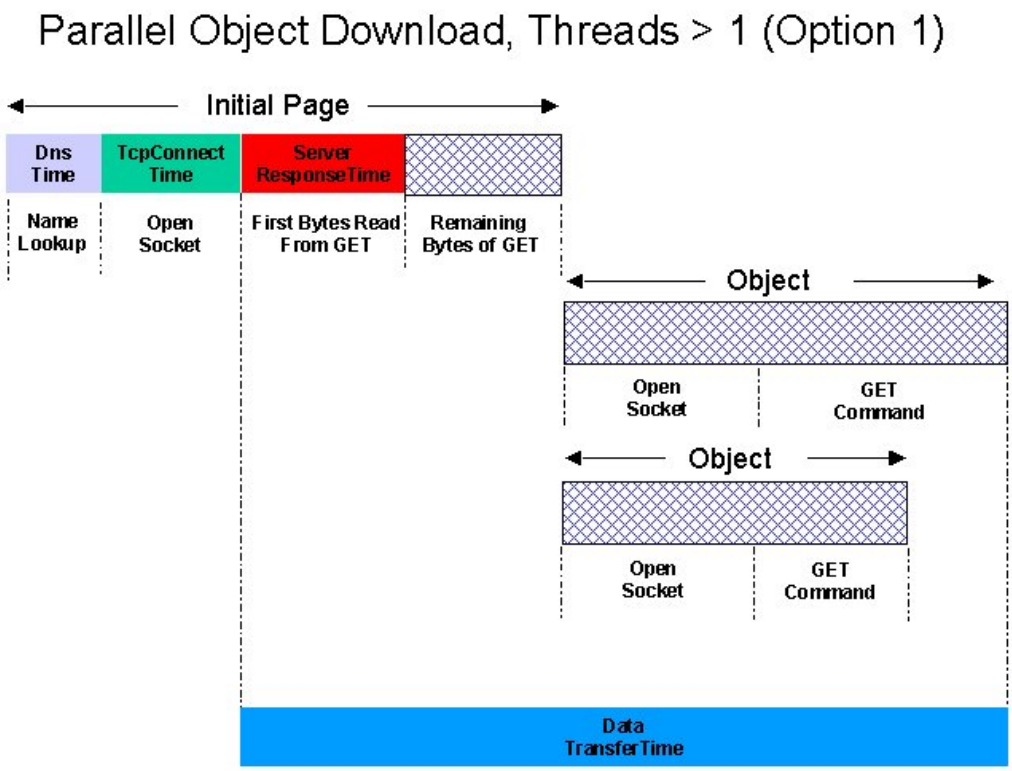


Figure 1

When run with DownLoadThreads greater than one, the test will download embedded objects in parallel using the specified number of threads. The default number of threads is four for new service models in Firehunter 3.3. This number of parallel threads will usually result in reasonable and accurate response time measurements without affecting scalability. However, it can be changed, if desired.

When run in the parallel download manner, the test calculates its measurements using a different algorithm (Figure 2). DnsTime, TcpConnectTime, and ServerResponseTime are all calculated in the same way as with DownLoadThreads equal to one. DataTransferTime and TotalResponseTime are calculated differently. Since the embedded objects are downloaded in parallel, DataTransferTime is **NOT** calculated by adding the time to get each embedded object, but rather is the amount of time from sending the request for the initial page until the last byte of the last embedded object is downloaded. DataTransferTime includes the ServerResponse-Time measured for the initial page. TotalResponseTime is calculated by summing DnsTime, TcpConnectTime, and DataTransferTime. ServerResponseTime is not explicitly added into TotalResponseTime because it is a component of DataTransferTime. AdditionalTcpConnect-Time is not meaningful when the embedded objects are downloaded in parallel, and it is not included in the calculation of DataTransferTime or TotalResponseTime. AdditionalTcp-ConnectTime is not monitored by default.

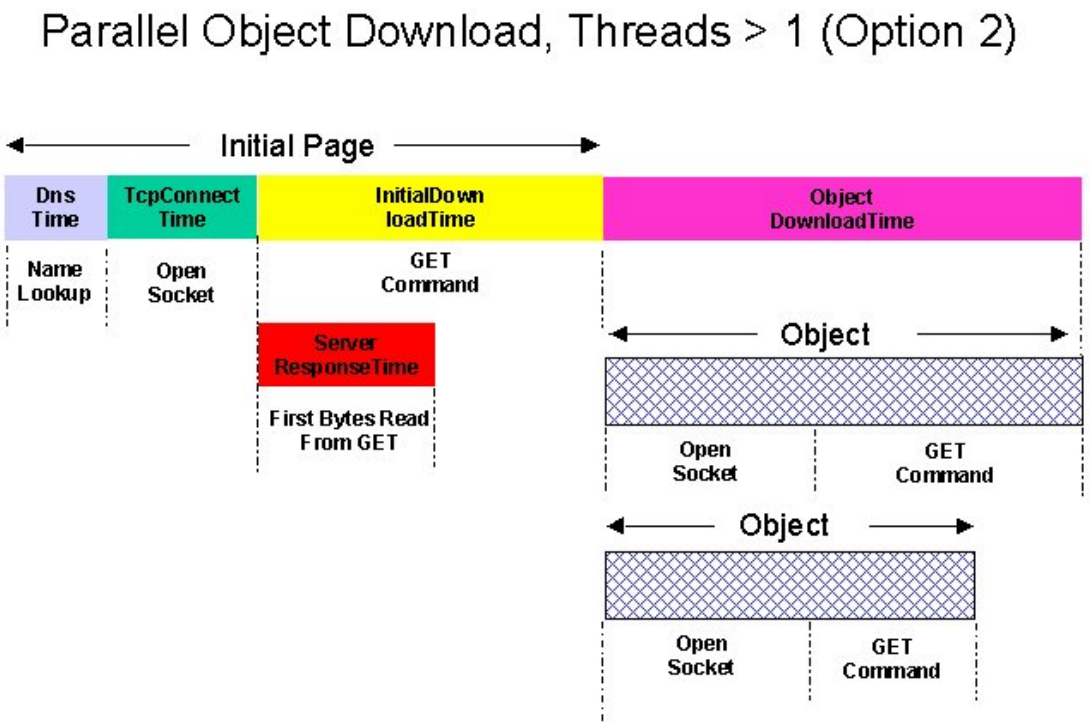
Note: Changing the DownLoadThreads from one to greater than one for an existing test will have implications to baselines and historical data since it alters the way DataTransferTime and TotalResponseTime are calculated.



TotalResponseTime = DnsTime + TcpConnectTime + DataTransferTime

Figure 2

For users that require more detail, there is a second optional set of measurements that break the DataTransferTime into two separate components, InitialDownloadTime and Object-DownloadTime (Figure 3). This can be a valuable option when the initial page is dynamically generated; for example, if data from a database is used in the initial page. It can also be important to separate these two components if many of the embedded objects come from a different server than the initial page. It is possible to configure the test to return all three measurements (DataTransferTime, InitialDownloadTime, and ObjectDownloadTime), but they are redundant.



TotalResponseTime = DnsTime + TcpConnectTime + DataTransferTime

Figure 3

There are two other measurements taken by the HTTP test that can be monitored, Test-Completed and EmbeddedObjectErrors (Figure 4).

As shown in this figure, TestCompleted refers only to the initial Web page. Since embedded objects can reside on other Web servers, errors retrieving them are reported with Embedded-ObjectErrors rather than TestCompleted. TestCompleted means that the initial Web page could be downloaded without error. In the case of a web page access that returned a 200 status (OK) and didn't include the MustHave string, the Availability measurement would be 100 percent because the Web server responded in an appropriate manner (that is, with a valid HTTP response). However TestCompleted would be zero percent because the downloaded Web page did not contain the MustHave string.

The EmbeddedObjectErrors is a count of the number of the embedded objects that could not be downloaded because of an error of some type (for example, 404 – Not Found).

Figure 4 shows how all the measurements are taken in parallel download mode. This helps illustrate the fact that the AdditionalTcpConnectTime is not a meaningful measurement when using parallel downloads. It is not monitored by default and will not help much with diagnosing problems. The two most appropriate options are to monitor DataTransferTime (Figure 2) or InitialDownloadTime and ObjectDownloadTime (Figure 3) when using DownLoadThreads greater than one. TestCompleted or EmbeddedObjectErrors can be combined with either of those options.

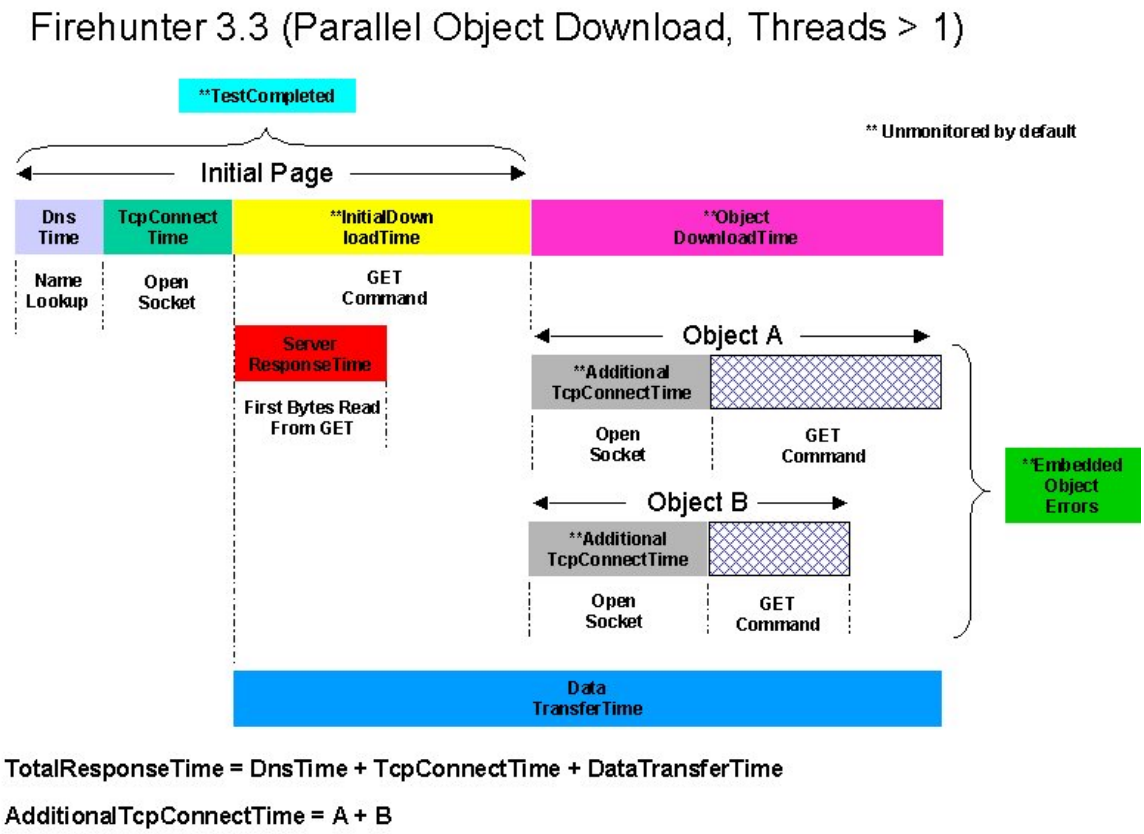


Figure 4

Summary

The Firehunter HTTP test provides a reliable, repeatable set of measurements to gauge the performance of a Web site. In addition, it provides drill-down capabilities in the form of separate component measurements of an HTTP request and response. While similar in behavior to a Web browser, the Firehunter HTTP test provides measurements that indicate the QoS being delivered by a Web service, and it does so in a highly scalable manner. The configuration options for caching and threaded operation can cause the test response times to more closely approximate a browser's response times if that is desired. But the purpose of the test is not to reproduce a specific "user experience," but to more accurately and consistently measure the QoS delivered. Another benefit is that by more closely following HTTP and HTML specifications, the test can quickly identify problems not found by browsers (such as the expiration dates of HTTP Cookies).

References

1 IETF RFC 2616 – Hypertext Transfer Protocol – HTTP/1.1, June 1999.

2 IETF RFC 2965 – HTTP State Management Mechanism, October 2000.

3 "Measurement and Management of Internet Services Using Agilent Firehunter," White Papers link at http://www.firehunter.com/techsupt, Agilent Technologies, 2000.

Appendix A – Measurement Definitions

- Availability - Indicates that the Web is available to service HTTP requests. For example, if the server responds with a valid status code that is less than or equal to 399, then the Availability will be 100 percent. If the server responds with unknown responses, failure to connect, or status codes that are greater than or equal to 400, then the Availability will be 0 percent.

- TotalResponseTime - Sum of DnsTime, TcpConnectTime, AdditionalTcpConnectTime (when threads equals one), and DataTransferTime. ServerResponseTime is not explicitly added into TotalResponseTime because it is a component of DataTransferTime.

- DnsTime - Time of DNS name lookup.

- TcpConnectTime - Time to establish the initial TCP connection with the service.

- AdditionalTcpConnectTime **** - Sum of all TCP connection times, excluding the initial TCP connection time. For example, AdditionalTcpConnectTime may include the connection time required to retrieve images for a Web page.

- ServerResponseTime - Time from when the request is sent until the first byte of the response is read.

- DataTransferTime - The time to load the Web page and all the embedded objects.

- DataTransferRate - Rate at which data from the Web page is returned.

- ValidPageContent **** - Indicates whether the content on the page meets the MustHave and FailsWith criteria.

- TestCompleted **** - Indicates either that the test completed without any errors (100 percent), or that the test encountered one or more errors and could not complete successfully (0 percent). This measurement applies to the initial Web page only, not to embedded objects.

- InitialDownLoadTime **** - Time to download the initial Web page.

- ObjectDownLoadTime **** - Time to download embedded objects (frames and images).

- EmbeddedObjectErrors **** - The number of embedded objects that had download errors.

**** Unmonitored by default.

Legal Notices