F5 TESTING METHODOLOGY

CREATING A ROBUST PERFORMANCE TESTING METHODOLOGY

CREATING A ROBUST PERFORMANCE TESTING METHODOLOGY

About This Document

This document is a collection of high-level performance testing guidelines, including how to design appropriate tests, how to spot biased tests, and how to apply this information to real world scenarios. To avoid confusion and limit scope, this document assumes that the performance of an Application Delivery Controller (such as the BIG-IP LTM) is being evaluated.

Table of Contents

CREATING A ROBUST PERFORMANCE TESTING METHODOLOGY

Introduction to Performance Testing

Successful performance testing requires a good deal of time and experience. It is a common mistake to assume testing the performance of Application Delivery Controllers (ADC's) is a simple task that can be conducted by any technical person with the appropriate equipment in a few weeks time. Performance testing requires a significant amount of forethought and diligence, as well as a high level of knowledge regarding the devices and protocols to be tested, and an intimate familiarity with the tools used to do the testing. It's easy to see how comprehensive and meaningful performance testing is largely relegated to a select group of experts, while many assume it should be no problem for any engineer to produce meaningful performance metrics after only a day or two of testing.

With that said, just about anyone can learn to conduct successful performance testing of ADC's. The most important factor (by far), is time. Having enough time to properly design, implement, run, tune and evaluate the test methodology, environment, devices and results is the key to success in performance testing.

In addition to time, there is one other indispensable factor when evaluating performance: forethought. Forethought implies thorough goal analysis, planning and proactive problem solving. Forethought would be regarded as more essential than time, except that a failure to think ahead can, to a degree, be compensated for if you have enough time.

The importance of these two key elements, time and forethought, will be reiterated throughout this document.

Designing Meaningful Tests, and Avoiding Bias

There are five steps to achieving success in any evaluation that includes performance.

1. Goals/Scoping
2. Methodology/Planning
3. Implementation/Execution
4. Validation/Evaluation
5. Results/Interpretation

Even one-off tests that can be completed in an hour to obtain a single metric still require these steps in order to be meaningful; the only thing that changes as the test size increases is the time spent on each step.

1. Goals/Scoping: "What do you want to achieve?"

Successful performance testing requires a clear set of narrowly defined objectives (goals). Success requires that goals be achieved, and to be achieved, goals must first be defined. Success achieving overly broad goals is not really success. In order to be useful goals must reflect what it is you really hope to learn from the testing. The majority of all incomplete test plans fail at this step by not taking into account the bigger picture: their goals.

Well designed goals have the following characteristics:

Problem/solution oriented
Each test should be clearly sourced from a problem, or set of related problems. The reason to conduct performance testing is to evaluate the capabilities of a device in order to address specific business and/or technical requirements. For example, if a specific business requirement is to support 1000 concurrent users, a test that determines the throughput capabilities of the device are unlikely to aid in that discovery. Testing goals that are not created to solve problems are more likely to fall prey to unfair or illogical bias, so having a clear problem/solution relationship when designing goals is key.

Describes success criteria for a single test or set of related tests
Each goal should state what problem is being solved, what specifically is being tested, and the definition of success for that goal.

Parity with the actual use case
Test goals should describe what the device being tested will ultimately be subjected to (i.e. they should be "real world"). If your goal is to enhance the speed of your web-based applications through the use of optimization technologies, a test of "TCP-only" connections/second without HTTP requests and responses is not helpful, and will tell you little or nothing about what to expect once the device is implemented. Goals that do not meet this requirement are likely the result of shortened time-lines, lack of forethought or expertise, and/or political objectives or ideologies.

CREATING A ROBUST PERFORMANCE TESTING METHODOLOGY

Designed independent of any vendor's capabilities
All stated goals should be created solely with the use case objectives in mind, without regard to the capabilities of any given vendor. Any testing designed based on or around a particular vendor's devices is inevitably going to be biased. The test goals should not define how the device solves the problem, only what the characteristics of success are.

The following are examples of good test goals:

- Number of successfully fulfilled HTTP requests for '/index.html' (5KB), using SSL (RC4-MD5) between the client and DUT (Device Under Test), and with HTTP compression enabled. At a minimum, the device should be capable of sustaining 10,000 HTTP requests per second from 5,000 different IP addresses (clients), with not more than 5 requests per TCP connection. Vendors will be ranked based on compression ratio.
- Total L2 throughput of requests for a web page (243KB) and 5 images (8KB each); the page must come from one group of servers and the images must come from another group. The device must support 2 Gbps of sustained throughput, and 10 second bursts of up to 3 Gbps. Vendors will be ranked based on time-to-last-byte (TTLB).
- Maximum concurrently open user connections for three different network types: dial-up (56Kbps, 100ms latency, 1% loss), DSL (768/384Kbps down/up, 20ms latency, 0% loss) and LAN (no WAN emulation), using three different reference file sizes on each network type: 128B, 2KB and 8KB. The device must scale linearly as the file size increases on each network type, and must maintain a time-to-last-byte (TTLB) of no more than 5 seconds. Vendors will be ranked based on TTLB averaged together for all sizes.
  **Note that this is actually 9 individual tests, but that they are still clearly defined.

The following are examples of ineffective test goals:

- L2 throughput, using a single if/then type rule.
- Number of connections/second.
- Concurrent users with <insert vendor-specific function/feature>, or equivalent, enabled.

Proper goal setting is the cornerstone of performance testing, and will make or break the rest of your efforts.

2. Methodology/Planning: "How do you plan to achieve it?"

Once you have a well thought out list of goals, designing a methodology becomes an almost automatic process. In general, the methodology should be as simple as possible while clearly articulating how to accomplish the stated goals. The methodology includes the specific individual tests to be run based on your goals, as well as the configuration settings of the devices being tested and the load testing tools (and any other devices necessary to run the tests). Complexity should be avoided in the methodology, while ensuring all parties involved understand it, and preferably agree with it (vendors, customer, etc.).

Having an established and agreed upon methodology is critical for any performance testing, because it is the pillar that all your conclusions are based on. Having the agreement of everyone involved in the testing prior to running the first test helps preclude anyone from disputing the results.

3. Implementation/Execution: "Doing what you planned"

The tests should be run as outlined in the methodology, in a transparent fashion providing visibility for everyone who has a stake in the testing. Performance testing generally takes more time than originally anticipated, since there are many separate devices involved in any one test. Inevitably, there are going to be issues with the test environment (switches, cables), load testing tools (configuration, bugs, performance limits), devices being tested (configuration, bugs) and possibly the methodology. Each vendor should be included in the testing to ensure that issues with their devices are resolved quickly, and that they are tuned for maximum performance.

Failing to seek cross-vendor input in any one of the steps up to and including this one is inevitably going to result in biased or inaccurate testing unless the tests are conducted by experts with years of experience in the field of performance testing Application Delivery Controllers.

CREATING A ROBUST PERFORMANCE TESTING METHODOLOGY

4. Validation/Evaluation: "Did you achieve your goals?"

The mantra of good performance testing is: "incorrect results are worse than no results at all." It is extremely important to validate and re-validate all goals and the methodology at each step, checking to make sure that the testing is comprehensive, fair and addresses the stated problems. Experience plays a big part in this step, since a seasoned performance tester will spot anomalies or inaccuracies more readily than someone unfamiliar with the intricacies of good performance testing.

When validating testing that has been done, it is important to look for the following three things:

- *Did the testing address all of the stated problems?*
- *Was the methodology executed as defined, and was it independent of any bias?*
- *Were there any anomalies or unexpected results/behavior?*

Any behavior or output that was unanticipated should be thoroughly researched and addressed before going on to draw conclusions from the results.

5. Results/Interpretation: "What does the output of the testing mean?"

This is one of the hardest parts of performance testing, and entire research papers have been written on the subject. When evaluating the results of any testing, it should always be in the context of the outlined goals and methodology. For example, making judgments on the number of concurrent connections during a test designed to demonstrate large file throughput is not useful, and could lead to incorrect conclusions.

Like the previous steps, interpretation of the results should be an inclusive endeavor. Everyone involved in the testing should have the opportunity to review and draw conclusions from the results. Any questionable results should be examined, and the corresponding tests re-run, if necessary. Consensus should be obtained by all parties before making any decisions based on the results of any performance testing.

Types of Performance Tests, and Why to Run Them

Terminology

Terminology is extremely important when doing performance testing. Not having an agreed upon set of definitions for commonly used terms can be disastrous, especially when they attempt to describe specific metrics. It's imperative that everyone involved in the testing understand and agree on the meaning of general terms used to define and discuss the testing. What follows is a list of commonly used terms, along with possible and recommended definitions.

TCP Three-Way Handshake (3WHS)

Definition
When dealing with TCP connections, a 3-way handshake is the process to open a connection between two hosts. In TCP, a 3-way handshake must occur before any data can be exchanged. For example, a web browser must first open a TCP connection before it can send an HTTP request.

A TCP 3-way handshake consists of a TCP/IP packet with the TCP SYN flag set from the originating host (client) destined for a remote host (server). The remote host responds with a TCP/IP packet with the TCP SYN and ACK flags set, and then the originating host sends a final ACK. These three steps (SYN, SYN/ACK, ACK) are referred to as a TCP three-way handshake (also known as 3-way handshake or 3WHS).

Connection (as in "Connections per second")

Possible definitions:
- New TCP connection attempt (SYN only)
- Full TCP connection establishment (3WHS)
- Full TCP connection establishment (3WHS) and HTTP request
- Full TCP connection establishment (3WHS) and HTTP request & response (complete HTTP transaction)
- Full TCP connection establishment (3WHS), HTTP request & response (complete HTTP transaction) and TCP close (FIN, ACK, FIN, ACK)
- Full TCP connection establishment (3WHS), HTTP request & response (complete HTTP transaction) and TCP abort (RST)

Recommended definition:
Full TCP connection establishment (3WHS), HTTP request & response (complete HTTP transaction) and TCP close (FIN, ACK, FIN, ACK).

Clarification:
Many Application Delivery Controller vendors use different definitions when they refer to a "connection" as part of a "Connection per second" metric. This difference in definitions leads to performance claims that are not comparable between vendors. Some of the less robust definitions can make a device look fantastic on paper, but this is misleading and does nothing to assist customers in making informed sizing decisions for typical environments. Additionally, a device that does extremely well in a SYN only "connections" test may perform very poorly in a "connections" test that uses the recommended definition. The recommended definition represents what will be typically seen in real-world deployment scenarios (i.e. real web browsers, servers, etc), and thus should be used for all performance testing of Application Delivery Controllers.

The recommended definition for a "connection" is sometimes referred to as an HTTP/1.0 connection (or HTTP/1.0 request), because original HTTP/1.0 implementations did not allow for more than 1 HTTP request per TCP connection. For many years HTTP/1.0 implementations have supported multiple HTTP requests per TCP connections, so this term is no longer accurate, but it's still commonly used.

It is important to note that this definition applies to the DUT. Some client applications (such as Internet Explorer) close connections with a RST instead of a proper TCP close much of the time. The purpose of this definition is to ensure that vendors are not taking connection handling "short cuts", and to clearly define a single grading scale and set of expectations for all vendors to follow.

Request (as in "Requests per second")

Possible Definitions:

- New TCP connection attempt (SYN only)
- HTTP request, independent of underlying TCP connection
- HTTP request and response, independent of underlying TCP connection
- Full TCP connection establishment (3WHS)
- Full TCP connection establishment (3WHS) and HTTP request
- Full TCP connection establishment (3WHS) and HTTP request & response (complete HTTP transaction)
- Full TCP connection establishment (3WHS), HTTP request & response (complete HTTP transaction) and TCP close (FIN, ACK, FIN, ACK)
- Full TCP connection establishment (3WHS), HTTP request & response (complete HTTP transaction) and TCP abort (RST)

Recommended definition:
HTTP request & response.

Clarification:
The terms "Connections per second" and "Requests per second" are often used interchangeably, but this should be avoided. It's prudent to clearly differentiate between the two because "connection" implies TCP (typically "L4" traffic, or an HTTP session with only 1 request), whereas "request" is more accurately defined as an entity operating at the application-layer (L7), and is often used to specify that there are multiple HTTP requests in a single TCP connection. For further clarification on the difference between L4 and L7, please see the ***L4 versus L7*** section below.

The recommended definition for a "request" is sometimes also referred to as an HTTP/1.1 request. This term is commonly used together with the term HTTP/1.0 connection (or request), as defined above in the Connection definition.

As hinted at above, it is desirable in many cases to open a single TCP connection and send multiple HTTP requests over it. The number of HTTP requests per TCP connection must be explicitly defined in your test. A convenient shorthand notation to indicate the number of HTTP requests per TCP connection is as follows: <#_req_per_client_conn>-<#_req_per_server_conn>, for example: "1-10" would mean a maximum of 1 request per client connection, and a max of 10 requests per server connection. In the case where the number of requests per connection is unlimited, "inf" (short for infinite) is a good placeholder. Common examples of this notation are: "1-1", "1-10", "1-inf", "10-inf", "inf-inf".

Not having consensus on the definitions of "connection" and "request" can impact all aspects of your testing, from design to interpretation of the results. It is imperative to have a common context for use in discussion and comparison, since at least one of these two terms will be relevant in every test you run.

Open Connection

Possible Definitions:

- Any TCP connection that is in the connection table of the DUT
- Any TCP connection that has finished the three-way handshake (3WHS) and is in the connection table of the DUT
- Any TCP connection that has finished the three-way handshake (3WHS), has processed at least one request & response, and is in the connection table of the DUT

CREATING A ROBUST PERFORMANCE TESTING METHODOLOGY

Recommended definition:
Any TCP connection that has finished the three-way handshake (3WHS), has processed at least one request and response, and is in the connection table of the device being tested.

Clarification:
This term will typically be used when designing concurrent user tests, and can have a tremendous impact on the final results based on which definition is used. When used in concurrent user tests, the goal of having a certain number of open connections is to simulate idle users. Alternate definitions may allow vendors to cheat this kind of test by using SYN cookies or other stateless technologies that fail to simulate real idle users. In order to be meaningful in real world deployments, the recommended definition should be used in all L4-L7 concurrent user tests.

Throughput

Possible Definitions:

- L2 throughput (total bits per second "on the wire")
- L4 throughput (payload data plus HTTP, TCP and IP headers; does not include L2 Ethernet headers)
- L7 throughput (payload data plus HTTP headers; does not include TCP, IP or L2 Ethernet headers)

Recommended definition:
L2 throughput (total bits per second "on the wire").

Clarification:
L2 throughput is the standard definition for essentially all networking devices and thus should be used to avoid confusion. The recommended definition corresponds to the same definition as network interfaces (i.e. 100Mbps, 1Gbps), and as such is the most useful for comparing tested performance versus maximum limits. The definition of this term has no impact on the final results, only how they are interpreted and compared.

It is important to note that some performance testing devices measure throughput at L7, and do not provide the option of measuring this metric at L2 or L4. Correspondingly, some equipment measures at L2, without the option of measuring at L4 or L7. Familiarity with this aspect of your load testing equipment is important when designing your tests.

Latency / Time-To-Last-Byte (TTLB)

Possible Definitions:

- Number of milliseconds for a single packet to be sent from server to client through the DUT
- Number of milliseconds to complete a single request & response, starting from the first SYN packet and ending on the last byte of response data received
- Number of milliseconds to complete a single request & response, starting from the first SYN packet and ending on the last FIN packet
- Number of milliseconds to complete a single request & response, starting from the first HTTP request packet and ending on the last byte of response data received
- Number of milliseconds to complete a single request & response, starting from the first HTTP request packet and ending on the last FIN packet
- Number of milliseconds to complete a web page (with embedded objects) request, starting from the first SYN packet and ending on the last byte of response data received
- Number of milliseconds to complete a web page (with embedded objects) request, starting from the first SYN packet and ending on the last FIN packet
- Number of milliseconds to complete a web page (with embedded objects) request, starting from the HTTP request and ending on the last byte of response data received
- Number of milliseconds to complete a web page (with embedded objects) request, starting from the HTTP request and ending on the last FIN packet

Recommended definition:
Number of milliseconds to complete a single request & response, starting from the first SYN packet and ending on the last byte of response data received.

CREATING A ROBUST PERFORMANCE TESTING METHODOLOGY

Clarification:
These two terms are commonly used interchangeably, which can be problematic if they have different definitions. Like all the previous terms, the recommended definition most closely matches the real world expectation, where the metric is judging the performance as it would impact a client (i.e. from the time they try to open a connection until they have enough information to display the data they have received). Similar to throughput, the definition of these terms have little impact on the final results, only the interpretation and comparison of them.

L4 versus L7

For several reasons, it's important to distinguish between tests of "L4" and tests of "L7". Many of the tests described below can be run to achieve the same metric at either layer. The difference between L4 and L7 tests can be summarized as:

- **L4**: Tests basic load balancing capabilities, and is defined by the ability to make a traffic management decision without inspecting/manipulating payload data.
- **L7**: Requires TCP multiplexing (many HTTP requests per TCP connection) and/or inspection/manipulation of payload data in order to make a traffic management decision, such as evaluating an HTTP request.

The definition for "connection" and "request" in the *Terminology* section has additional details and further clarifications on the similarities and differences of L4 and L7.

Basic Tests

Following is a list of basic tests, along with what the test measures. Each of these tests is designed to test a specific maximum capability for the purposes of sizing or comparison. The majority of all performance tests are derivatives of these basic tests.

Each of the basic tests can be either L4 or L7 tests (as defined in the preceding ***L4 versus L7*** section).

Connections/Requests per Second

Usually a series of tests with different response web page sizes, this test is used to measure the total number of TCP connections (users) or HTTP requests (transactions) a device can process over a specified time period (generally one second). This translates directly into the maximum number of users/transactions the device can realistically support at any given moment.

As described previously, the primary difference between connections and requests is that connections are testing the TCP capabilities, whereas requests evaluate the HTTP implementation.

These tests are sometimes run in parallel with a concurrent connections test.

Concurrent Connections

This test determines the maximum number of open connections (users) a device can support. This is accomplished by opening as many connections as possible and leaving them idle (or "open"). The intent is to completely fill the connection table of the DUT (typically limited by memory) in order to determine the maximum number of connections it can support, usually in combination with the TTLB of a few requests at a given level of concurrency.

Variations of this test include opening and leaving idle a set number of connections, and then proceeding to run a connections/second or requests/second test. This test simulates user inactivity or "think time", which is often appropriate since in common real world environments the majority of users will be idle or "thinking" at any given time.

Throughput

Throughput measures the total amount of data the device is capable of processing. This is generally accomplished by running a connections/second or requests/second test using a large response size (512KB or 1MB, for example) that provides the best overhead / transmission ratio. As the response size is increased in connections/second and requests/second tests the throughput will generally plateau at a point where the overhead in opening and closing connections no longer impacts the transmission of data. Thus, it could be said that throughput is simply a connections/second or requests/second test with a file size large enough to test the maximum data throughput, or "bits in, bits out", capability of the device being tested.

It's important to note that throughput is always measured using connections or requests as defined above. This is not a test of the maximum L2 switch throughput; this is a test of the maximum throughput of L4-L7 features as measured at L2.

CREATING A ROBUST PERFORMANCE TESTING METHODOLOGY

Advanced Tests

Using the basic tests as a basis, you can then move on to testing specific features or capabilities. The data gathered from your basic tests will generally be used as a baseline, so that you can make direct and insightful determinations on the impact of enabling/disabling specific configuration options or features.

The advanced tests described here are essentially just like the basic tests described above, with the only difference being that a specific option or set of options is changed on the DUT or on the load testing equipment. For example, an "SSL TPS" test is just a connections/second test that has SSL enabled. Most tests are labeled based on what feature/function is being tested, as a product of the way people think about these features and the implementation used by industry-standard load testing equipment (such as Spirent and Ixia). Thus, two otherwise identical tests with only one variation may have completely different names (i.e. "throughput" can become "SSL bulk crypto" when testing with SSL termination enabled, even though it's typically just called "SSL Throughput"). It's helpful to think of basic tests as baselines, and advanced tests as variations used to test specific features or functions.

Generally speaking, all advanced tests will be L7.

SSL

Measures the benefit of SSL termination at the DUT, or "SSL offload". SSL tests can be based on the requests/second or throughput basic tests. In addition to enabling SSL termination, it's common to have some tests that vary SSL-specific parameters, such as the number of allowed SSL Session ID reuses. Common parameters for SSL Session ID reuse are:

- No SSL Session ID reuse.
- 9 SSL Session ID reuses (total of 10 Session ID uses).

SSL tests that focus on connections per second are typically called "SSL TPS" tests (SSL transactions per second), and SSL tests that focus on throughput are typically called "SSL throughput", "bulk SSL throughput", or "bulk crypto throughput".

Caching

Measures the benefit of caching response content on the DUT, thereby offloading connections from the back-end servers and responding to client requests more rapidly. In addition to enabling caching, it's common to vary the percentage of cacheable content being returned by the back-end servers in order to derive more information regarding a vendor's caching implementation. Common parameters for percentage of cacheable content are:

- 0% Cacheable (Completely dynamic content)
- 50% Cacheable
- 100% Cacheable (Completely static content)

Each device being tested may have its own caching configuration options, including what content to cache and for how long, which should be left at default settings unless tuned by the vendor. The tuning and the justification for the tuning should be documented in the test methodology.

This test also has a unique metric, "ratio of hits", which measures the number of client requests that are served out of the device's cache rather than from a back-end server over the total number of requests (cached requests / total requests).

Compression

Measures the benefit of offloading HTTP compression to the DUT and/or the benefit of reducing the amount of data that must be transmitted. In addition to simply enabling compression, it's common to vary the compressibility of the content being returned by the back-end servers in order to derive more information regarding a vendor's compression implementation. Common parameters for percentage of compressible content are:

- 0-1% Compressible (Truly random or encrypted data. 0-1% ratio can be achieved against all compression algorithms)
- 50% Compressible
- 99-100% Compressible (Duplicate data, such as a file filled with a repeating string or single character)

Each device being tested may have its own compression configuration options, such as the compression algorithm or algorithm optimization settings, which should be left at default settings unless tuned by the vendor (what and why must be documented in the methodology).

This test has a unique metric, "ratio of compression", which measures the total L7 data transferred to the client (does not include L2 Ethernet, IP or TCP headers) over the total L7 data sent from the servers (compressed data / total data).

CREATING A ROBUST PERFORMANCE TESTING METHODOLOGY

Mixed

Mixed tests are the combination of a number of different types of tests into one traffic pattern. The idea is to simulate a wide range of end user activity in order to test multiple areas of a DUT simultaneously. This provides a better example of how the DUT might fare in the expected production environment. Mixed tests are often the best at finding limitations and/or bugs in the DUT's.

Vendor-Specific

This category of tests is for specifically testing any unique features of a particular vendor which are expected to be used in the eventual production deployment. Similar to the other feature-based advanced tests (SSL, Caching, Compression, etc.), the goal is to isolate the effect of enabling/disabling the feature in question.

Deployment-Specific

All the tests described above (both basic and advanced) are baseline tests used to determine the maximum performance capabilities of certain aspects of the devices being evaluated. Through a comprehensive run of the previous tests, you can obtain a performance profile of how a device will perform under certain conditions.

The goal of running tests in this category is to cover any nuances of the deployment environment which are not addressed with any of the previous tests. Typically, this is performance evaluation of advanced L7 intelligence.

Keep in mind that while the majority of the previous tests (both basic and advanced) are focused on establishing the maximum performance of a feature or capability, tests in this category will generally be focused on the maximum performance of a deployment scenario. Thus, tests in this category can be mistaken for mixed tests. The difference is that in mixed tests the goal is defined as "maximum <metric> with <selected traffic mix>", whereas deployment-specific tests are defined as "meet <specific conditions> while accomplishing <specific tasks>" in deployment-specific tests. It's important to make this distinction while designing tests, since the deployment-specific definition should define the deployment scenario (instead of being a generic test) and will allow vendors a greater deal of latitude in tuning their devices, since there are a number of conditions that need to be met rather than a single metric. The most flexible Application Delivery Controllers will really have an opportunity to shine in these tests.

It's extremely important to have the "specific conditions" and "specific tasks" clearly defined, and to evaluate all vendors using identical tests.

Spotting Biased or Broken Testing

Here is a list of things to look out for in order to spot biased or broken testing.

- Data "piggy backed" on the three-way handshake (SYN, SYN/ACK, PSH/ACK instead of SYN, SYN/ACK, ACK, PSH/ACK). Real operating system TCP stacks do not use "piggy backing".
- Connections closing with RST instead of a proper four-way TCP close (FIN, ACK, FIN, ACK). Either may be acceptable, but which is used should be monitored for consistency between the devices being tested.
- Lack of functional equivalence. There may be multiple ways of accomplishing the desired result on any given vendor's devices. Always use the simplest function on any given device, and rely on vendor recommendations and guidance.
- Mandatory inclusion of unique features in non-vendor-specific tests. If there is a desire to test a feature of one vendor's devices that is not shared by the rest, this should be addressed with a completely separate and independent test. Requiring the use of unique features in shared tests eliminates any functional equivalence, negating the results and creating bias. This, of course, does not preclude vendors from using unique features to better meet the test requirements if they choose to do so.
- Tests that have been "dumbed down" to the lowest common denominator (for example, a test that defines the solution instead of the problem; "using L7 rules" instead of "based on <special piece of data>"). Each vendor should be able to choose how best to meet the test criteria, which should be designed with the actual use case in mind (as covered in the *Goals/Scoping* section). Vendors with more capabilities should not be restricted to only using certain features because of limitations in other vendor's devices – the test case should only define the problem to be solved, not how to solve it.
- Requirements that have been defined based on a particular vendor's product, or a vendor's devices being used to build the test configurations. The goals and methodology of the testing should be designed independent of any particular vendor solution (as covered in the ***Goals/Scoping*** section), in order to avoid stifling the creative problem solving capabilities of feature-rich devices.

CREATING A ROBUST PERFORMANCE TESTING METHODOLOGY

- Beware of case insensitivity. Some vendors do not support case insensitive inspection, in which case all testing should be case sensitive. If case insensitivity is required, devices that do not support this will need to check for every possible case (i.e. FOO, FOo, FoO, Foo, fOO, fOo, foO, foo) in order to achieve functional parity.
- Unclear or undefined grading scale or selective grading. The expected output for each test should be clearly defined. As part of the goals and methodology, each vendor should know exactly what is being measured in each test. All devices should be graded on the same criteria for every test.
- Metrics that are recorded from the DUT. All measurements should be taken on the test equipment (load generation tools, switches, etc), and not from the DUT. As an unbiased tester, you must assume all DUT's report inaccurate performance metrics (CPU usage, connections per second, compression ratio, etc).
- All vendors should have equal opportunity to tune their devices for each test scenario, just as they would for a real deployment.

I Live In The Real World and Not A Lab: What Do I Do?

All the information up until this point is meant to describe the ideal performance testing case, where there is enough time allotted and all vendors are allowed equal opportunity to provide feedback and tune their devices. However, it's well known that this rarely happens. In most cases, there is not enough time allocated, or the engineers conducting the tests do not have the resources or expertise necessary to do proper testing, or the test engineers are unwilling to accept vendor input on the testing, or some of the above, or all of the above.

What should be done in these circumstances? First and foremost, it's important to use what you have already learned about properly executed performance testing to persuade any dissenters and hopefully change course and do what's necessary to conduct proper testing. In the absence of a change of heart on the part of the dissenters, the following is a list of suggestions for dealing with testing being done on a short time scale under sub-optimal conditions.

Know the products

Set aside a day or two and run tests "back to back" using only the load testing equipment (no switches or DUTs). Change settings and see the effect. Then add in your L2 switch and run some throughput tests. Bring the vendor of your load testing equipment onsite to help out. If you know you'll be running similar tests in the future but can't dedicate the test infrastructure for this purpose, make sure you have copies of the configurations once you're satisfied with the configuration. Using known "good" configurations for future tests can help reduce the time required to reconstruct the test environment.

At the same time you're familiarizing yourself with the test equipment and making sure it works as needed, ask the vendors to configure their own devices for the tests you want to run (have them bring the devices onsite pre-configured). Make sure you'll have vendor representatives' onsite during setup and execution of your testing.

Watch out for shortcuts

Some common tactics employed to compensate for a lack of time or forethought include, but are not limited to:

- Using basic tests with no relevance to the expected use once the product is deployed. Remember: if the tests that are run do not simulate real clients and servers, and do not test expected uses of the product: they are not worth running! It's better to run fewer tests and get good results.
- Designing and implementing the first round of tests using one vendor's equipment.
- Asking the rest of the vendors to setup their devices for testing in a short time frame, after the first vendor has had the benefit of a longer time frame to refine their configurations.
- Employing L2-L3 performance equipment and/or tests, instead of proper L4-L7 test gear and testing methodology. Remember: if the tests that are run do not simulate real clients and servers, and do not test expected uses of the product: they are not worth running! It's better to run fewer tests and get results that are meaningful.
- Trying to implement the tests without the proper training and experience, and potentially using rented performance testing gear that they are not familiar with.

Remember that if the tests that are run do not simulate real clients and servers, and do not test expected uses of the product, they are not worth running. It is better to run fewer tests and get meaningful results.

CREATING A ROBUST PERFORMANCE TESTING METHODOLOGY

Leverage known information
Have previous performance testing reports (like the Broadband Report – http://www.f5.com/products/pdfs/BBTV9_Performance_Report.pdf) and vendor marketing materials on-hand and know what their contents are. Having information readily available to dispute inflated or suspect results can save time by identifying when re-testing may be appropriate before moving on to other tests.

Example Scenario

In this section, we outline an example customer scenario and matching methodology using the principles described in the rest of this document.

Important:
This is a completely fictitious example, meant to demonstrate how to translate customer requirements into a methodology based on the process outlined in the rest of this document. The customer requirements are completely contrived, and are not based on any real use case. The example methodology is only an example, and is not a recommended test plan. Any performance testing methodology you develop should be based on the stated goals/objectives of you or your customer; do not simply copy and paste this scenario.

Customer Problem
The customer has a farm of 20 web servers (5 image/Flash servers and 15 page servers) delivering dynamic content from 5 database servers for a partner portal that averages 50,000 simultaneously active users and 4 Gbps of L2 throughput. The average total page size is about 165KB, with roughly 25 in-line images and a Shockwave Flash banner. They would like to offload server SSL and compression, as well as provide caching for all static content. There are three vendors being evaluated, and the customer would like to do performance testing to compare the offload capabilities, as well as test some feature sizing comparisons.

Customer Requirements
The customer will pick the vendor with the best performance/price ratio that can sustain a concurrent user and throughput load 30% greater than their current averages. All users will be coming from some type of high-speed Internet connection, so the customer does not care about WAN optimization or emulation, but would like to maintain a reasonable response time (less than 5 seconds per complete page load). They must log all client IP addresses, but don't care if the IP is preserved or is delivered in an HTTP header. Anything that can be done to offload server-side processing is welcome, as long as it doesn't require any changes to the clients. All vendors have demonstrated database persistence for connections from the web servers, so they only desire to test the client experience in regards to the web servers (this is their primary bottleneck). Since users must be authenticated and the content is completely dynamic based on who they are, users must visit the same web server for the duration of each session; the customer would like to use their existing authentication token (an HTTP cookie) for this.

Example Methodology

Global Definitions
For this methodology, these terms will have the following definitions:

Test Page
The predetermined content which will be used to simulate the average page on the deployment site. The Test Page consists of:

- 1x 20KB page with included CSS and JavaScript
- 10x 1KB images
- 5x 2KB images
- 5x 5KB images
- 5x 10KB images
- 1x 50KB Shockwave Flash

Servers
There will be one group of 5 image/Flash servers and another group of 15 page servers in every test using the Test Page. In all other tests, only the 5 image/Flash servers will be used (no page servers).

Full HTTP Headers
Includes HTTP headers that simulate Internet Explorer 7.0 in client requests, and simulate Apache 2.x in server responses.

CREATING A ROBUST PERFORMANCE TESTING METHODOLOGY

Connection
An HTTP/1.1 request with only a single request per TCP connection from the client's perspective. HTTP requests and responses must include full HTTP headers and use standard connection termination (four-way TCP close). This must be a "real world" example, so the DUT's are not permitted to use any non-standard TCP or HTTP optimization techniques, such as including data in the three-way handshake or closing connections with a RST.

Request
An HTTP/1.1 request with full HTTP headers.

Open Connection
A connection that has finished the three-way handshake, completed one request and response, and is in the connection table of the DUT.

Time-To-Last-Byte (TTLB)
Number of milliseconds to fulfill all requests required to download the complete Test Page and all embedded objects, starting from the first SYN packet and ending on the last byte of response data.

Failure
Once a single client request does not succeed, or the test parameters are violated, the device will be considered "failed" for the rest of the test.

Test 1: Throughput

Description
Maximum Gbps of traffic the device can sustain with at least 65,000 concurrent connections requesting a 1,000,000 byte test file.

Expectation
Ability to sustain greater than 4 Gbps of load balanced throughput, measured at L2. Vendors will be ranked based on L2 throughput, which will be measured from the 2nd to the 4th minute of the final steady-state.

Ramp Up/Down
Ramp to 65,000 users at a rate of 5,000 users/10 seconds, and then maintain steady-state of 65,000 users for 5 minutes.

Configuration Options

SSL:	off
Caching:	off
Compression:	off

Iterations
This test has only one iteration.

Test 2: Concurrent Connections

Description
Maximum number of concurrent open connections the device can successfully sustain. The initial request will be for a 128B file.

Expectation
Ability to support greater than 100,000 concurrent open connections. Vendors will be ranked based on number of concurrent open connections, which will be measured for the last 30-90 seconds before failure.

Ramp Up/Down
Ramp to failure at a rate of 5,000 users/10 seconds with 1 minute steady-states every 20,000 users.

Configuration Options

SSL:	off
Caching:	off
Compression:	off

Iterations
This test has only one iteration.

CREATING A ROBUST PERFORMANCE TESTING METHODOLOGY

Test 3: Requests per Second

Description
Maximum number of requests/second the device can successfully sustain with users requesting the Test Page and its embedded objects.

Expectation
Ability to sustain greater than 1,000 requests/second with a TTLB of no more than 5 seconds. Vendors will be ranked based on number of requests/second, which will be measured for the last 30-90 seconds before failure.

Ramp Up/Down
Ramp to failure at a rate of 5,000 users/10 seconds with 1 minute steady-states every 20,000 users.

Configuration Options

SSL: off
Caching: off
Compression: off

Iterations:

This test has only one iteration.

Test 4: SSL

Description
Maximum number of SSL terminated requests/second the device can successfully sustain with users requesting the Test Page and its embedded objects.

Expectation
Ability to sustain greater than 1,000 requests/second with a TTLB of no more than 5 seconds. Vendors will be ranked based on number of requests/second, which will be measured for the last 30-90 seconds before failure.

Ramp Up/Down
Ramp to failure at a rate of 5,000 users/10 seconds with 1 minute steady-states every 20,000 users.

Configuration Options

SSL: on
Caching: off
Compression: off

Iterations

One run with no SSL Session ID reuse.
One run with 9 SSL Session ID reuses (total of 10 uses).

Test 5: Caching

Description
Maximum number of caching enabled requests/second the device can successfully sustain with users requesting the Test Page and its embedded objects.

Expectation
Ability to sustain greater than 3/5 ratio of hits and 1,000 requests/second with a TTLB of no more than 5 seconds. Vendors will be ranked based on number of requests/second and caching ratio, which will be measured for the last 30-90 seconds before failure.

Ramp Up/Down
Ramp to failure at a rate of 5,000 users/10 seconds with 1 minute steady-states every 20,000 users.

Configuration Options

SSL: off
Caching: on
Compression: off

CREATING A ROBUST PERFORMANCE TESTING METHODOLOGY

Iterations

This test has only one iteration.

Test 6: Compression

Description

Maximum number of compression enabled requests/second the device can successfully sustain with users requesting the Test Page and its embedded objects.

Expectation

Ability to sustain greater than 50% compression of text content (must not attempt to compress images or Flash) and 1,000 requests/second with a TTLB of no more than 5 seconds. Vendors will be ranked based on number of requests/second and compression percentage, which will be measured for the last 30-90 seconds before failure.

Ramp Up/Down

Ramp to failure at a rate of 5,000 users/10 seconds with 1 minute steady-states every 20,000 users.

Configuration Options

SSL:	off
Caching:	off
Compression:	on

Iterations

This test has only one iteration.

Test 7: Total System Performance

Description

Maximum number of requests/second the device can successfully sustain with users requesting the Test Page and its embedded objects. Clients must be directed to the same server for the duration of each session based on the HTTP cookie they receive from the application, and their IP address should either be preserved or inserted as an HTTP header.

Expectation

Ability to sustain greater than 1,000 requests/second with a TTLB of no more than 5 seconds. Vendors will be ranked based on requests/second, which will be measured from the 2nd to the 4th minute of the final steady-state.

Ramp Up/Down

Ramp to 25,000 concurrent open connections at a rate of 5,000 users/10 seconds, then ramp requests to 40,000 at a rate of 5,000 users/10 seconds with a 1 minute steady-state at 20,000 users and a 5 minute steady-state at 40,000 users.

Configuration Options

SSL:	on (no SSL Session ID reuse)
Caching:	on
Compression:	on

Iterations

This test will be run 3 times back-to-back (no reboots), and the results will be averaged.