

# **Avalanche Load Generation:**

## How to Improve your Rate-based Tests

---

John Kenney

Product Feature White Paper

April, 2005



## Abstract

By allowing you to simulate interactions that are as demanding and volatile as what happens in the real world, Avalanche produces truly meaningful test data. This capability carries with it complexity.

This white paper explains a) how load is generated, b) why there may be discrepancies between the desired load and actual load you test, and c) what you can do to eliminate these issues and take full advantage of Avalanche's rich set of customizable tests.

## About the Author

Dr. John Kenney is Director of Engineering for the Avalanche Product Line at Spirent Communications. Instrumental in the development of the Avalanche and Reflector network testing appliances, he currently heads the design team working on the continuing refinement and quality assurance of the Avalanche product line. A ten-year industry veteran, John served as Chief Technology Officer and Vice President of Engineering at Caw Networks, and was one of the founders of ePatterns, where he served as both Director of Engineering and Director of Marketing. His distinguished background at Stanford University is highlighted by nine years as a research associate and research assistant, a Master's degree in Computer Science, and a doctorate in Electrical Engineering

### **Spirent Communications**

26750 Agoura Road  
Calabasas Hills, CA  
91302 USA  
e: [enterprise@spirentcom.com](mailto:enterprise@spirentcom.com)

#### **Sales Contacts:**

##### **North America**

+1 800-927-2660

##### **Europe, Middle East, Africa**

+33-1-6137-2250

##### **Asia Pacific**

+852-2166-8382

##### **All Other Regions**

+1 818-676-2683

[www.spirentcom.com/enterprise](http://www.spirentcom.com/enterprise)

Copyright 2005 by Spirent Communications, Inc. All rights reserved.

# Table of Contents

<b>1. Introduction.....</b>	<b>1</b>
1.1 Some Definitions .....	1
1.2 Traffic Generation .....	3
1.3 The User's Perspective .....	4
<b>2. Problems with Consistency .....</b>	<b>4</b>
2.1 Reporting.....	5
2.2 Physical Speed and Capacity.....	5
2.3 Animating Users.....	6
2.4 Graphical Representation .....	6
<b>3. Understanding the Problem Mathematically .....</b>	<b>7</b>
<b>4. Tips and Tricks .....</b>	<b>9</b>
4.1 Planning with Limits in Mind .....	10
4.2 Areas to Troubleshoot .....	10
4.2.1 User Profile .....	10
4.2.2 Load Profile .....	11
4.2.3 Action Lists .....	11
<b>5. Conclusion .....</b>	<b>12</b>

## 1. Introduction

It could be Cisco testing a firewall or Costco.com testing their Web site: companies who want dependable quality assurance use Spirent. Instead of just blasting devices with traffic, Spirent's load generation algorithms allow you to tailor your tests to reflect what users are really going to put your devices through. Being able to mimic real-world situations provides the most sophisticated—and the most useful—information about how your equipment or Web site performs.<sup>1</sup>

This white paper explains how Avalanche generates load, why variations in the generated load exist, and what tricks of the trade you can use to improve your rate-based tests.

### 1.1 Some Definitions

In Illustration 1, the user has decided to test how a firewall handles 20,000 TCP connections. This type of measurement is known as a **concurrent metric** because the 20,000 connections are all happening at the same time and new connections only open to replace connections that have closed. In other words, the number of connections will stay constant for the duration of the test.<sup>2</sup> Other examples of concurrent metrics would be testing a set number of simulated users or HTTP transactions.

#### Open Connections Capacity Test — 20,000 Open Connections

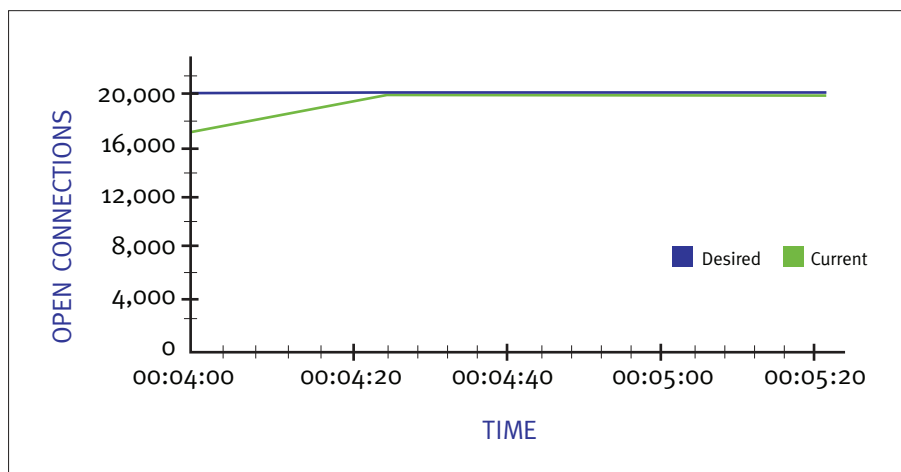


Illustration 1: Concurrent Metrics Testing 20,000 Connections

<sup>1</sup> This white paper's focus is on realistic testing, but Spirent's tools allow you to isolate and verify faults with unrealistic tests, too. These tests are predictable, repeatable, and laboratory-precise.

<sup>2</sup> In real life, connections close for a variety of reasons (for example, if a Web page has finished downloading). You can keep connections open indefinitely if you are performing unrealistic tests.

Illustration 2 tells a different story. It demonstrates a rate-based metric—in this case, the user’s test is to add 20,000 connections each second. Avalanche works to keep these rates accurate, but there are circumstances under which Avalanche seems to add 19,900 connections/sec or 20,100 connections/sec instead. In Section 3, we’ll be describing more about why it isn’t possible to always add exactly 20,000 connections/sec. We’ll also be giving some tips to get very close.<sup>3</sup>

### Connections/Sec Rate Test — 20,000 Connections/Sec

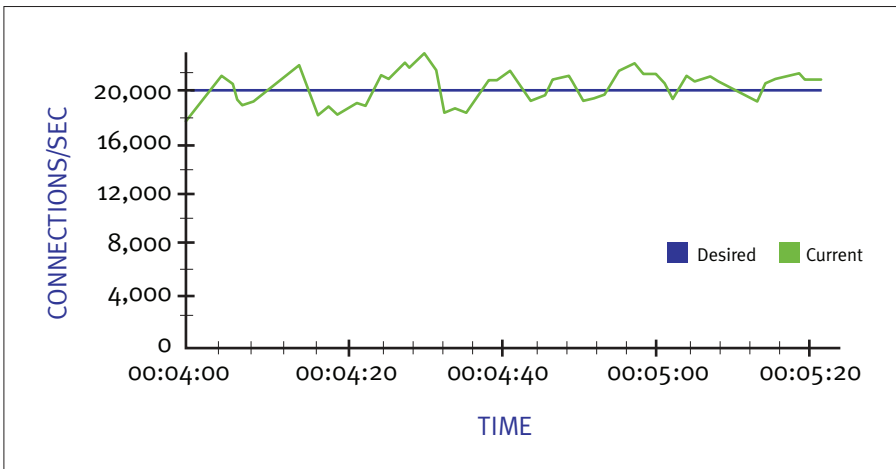


Illustration 2: Rate-based Metrics Testing 20,000 Connections/Sec

To understand the difference between concurrent metrics and rate-based metrics, imagine load testing a restaurant. If we want to stress test Mel’s Diner, we could see when failure occurs if there are always 100 customers. This would be a concurrent metric—people come in to eat, finish and leave, but new customers immediately take their places, keeping the restaurant packed at a constant 100. Maybe it takes two days before the kitchen overheats and the cooks collapse, maybe it takes two hours. Before any catastrophic failure, a stress test will usually first generate a slow-down. Mel’s Diner could handle 100 customers for a while, but food would take progressively longer to arrive and the quality would probably deteriorate.

We could also run a stress test with a rate-based metric. This kind of test would start Mel’s Diner off at 6 a.m. with zero customers. Every hour would bring 30 customers (in other words, one customer every two minutes).

With every passing moment, let’s assume there are more customers arriving than there are customers finishing up and leaving. As time goes on, the diner gets overloaded; the number of customers waiting for service increases and the time each customer spends waiting also increases.

<sup>3</sup> Please note that “**load specification**” is really just another name for these **metrics**.

Yet more customers keep crowding in each minute. By mid-afternoon there are so many customers waiting that the waiters can't move and no one eats—Mel's Diner has reached its failure point.<sup>4</sup>

## 1.2 Traffic Generation

In approaching the problem of simulating real-world traffic, Spirent engineers had to contend with a seeming contradiction. On the one hand, real-world traffic is unpredictable and bursty; on the other, a good testing tool is predictable and smooth. Avalanche solves this predicament through the use of a Load Controller and a Load Engine.

Measurements, often based on the state machines of the various protocols in the traffic, are fed into the Load Controller. The Load Controller is in charge of starting and stopping traffic, as well as adding and removing it. The instructions from the **Load Controller** go into the **Load Engine**. It's the Load Engine that generates the traffic. The Load Engine also passes feedback measurements to the Load Controller.

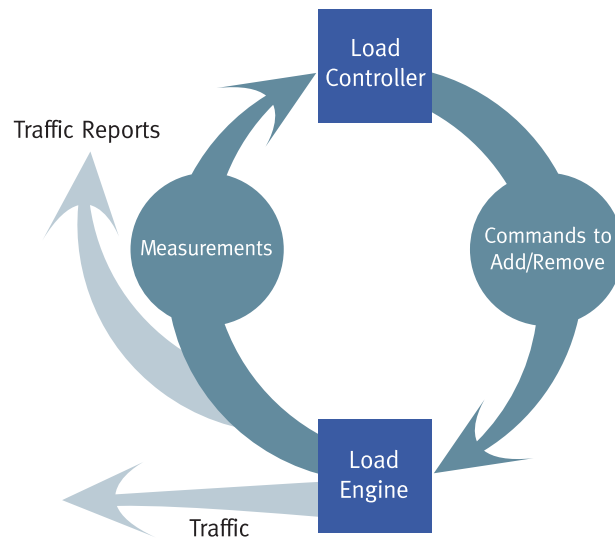


Illustration 3: The Load Generation Architecture

The feedback cycle between the Load Controller and the Load Engine (Illustration 3) makes Avalanche a self-regulating traffic system. It's important to realize that the Load Controller keeps a history of measurements and uses that history to determine how much traffic to add. Though the load is generated very quickly, the system must take into account that it does take some time to push out the traffic. Similarly, it does take some small fraction of time to look at the measurements.

<sup>4</sup> However, the performance of the diner degraded before it completely failed—customers had to wait increasingly long times before being served.

Though the combination of this lag is on the order of milliseconds, events can happen during this time that will have an impact on the consistency of the generated load.<sup>5</sup>

### 1.3 The User's Perspective

Avalanche generates simulated Internet traffic that factors in real-world characteristics such as connection speed, packet loss, browser emulation, think-time and aborted transactions. Its model for simulation is centered upon the concept of the **simulated user**, which is a combination of a User Profile and a corresponding Action List. The User Profile describes the user's behavior. For example, User Profiles specify the network characteristics of the users including their link speed, network latency and IP address range. The Action List includes the host name or host address and the "action" to perform—retrieve a Web page, send an email, play a media file, etc. A simulated user is born, walks the Action List (following the behavior indicated by its User Profile), and dies. All load is generated this way—by creating simulated users, having them perform a specific set of tasks and then disappear. To generate more load, the system adds more simulated users. The number of simulated users generated will satisfy whatever type of load you have specified, whether you specified users, connections or transactions.

One simulated user does not equal a predictable amount of traffic. Rather, the traffic generated is heavily dependent upon the performance of the device or system under testing. As the device or system slows down or fails, existing simulated users have a harder time making it through the Action List, so fewer simulated users die. Since fewer simulated users are dying, concurrent metrics will give birth to few new ones. So the traffic generated is reduced as the device fails.

This is a more accurate way to test load than just hitting one URL a million times, since that is not what happens in the real world. That simple type of load testing, however, is far easier to control and attenuate. The key difference with Avalanche is that the atomic metric of load generation is the user session, which plays out over a non-trivial amount of time, thus simulating real-world conditions but not allowing instantaneous changes to generated load levels.

A realistic test allows you to examine your device or system at work under a variety of conditions—not just one or two. Testing a variety of conditions translates into testing scenarios that have a multiple variables—for example, both how fast the Device Under Test (DUT) can initiate a TCP connection, as well as how fast the DUT can process an action from the Action List and return an acknowledgment. Simulated users are at the heart of Avalanche because the user perspective allows tests to accurately synthesize all the independent variables that are involved in realistic testing.

## 2. Problems with Inconsistency

The previous sections have defined our basic terms; now that these pieces are in place, we can talk about why sometimes the actual load being tested does not match the desired load that you specified.

<sup>5</sup> In addition to measuring traffic, making load decisions and adding users, the processor is also responsible for generating reports and handling the users that are already in progress. These last two tasks are more processor-intensive than the other three. Since there is usually only a single processor doing all of this work, we can understand the lag by looking at required processor cycles.

One of your first steps towards eliminating inconsistencies is determining the root of the problem. Some limitations are due to the device or system that you're testing; others are limitations in Avalanche. In addition to the "desired" and "actual" load results, pay attention to Avalanche's attempts to get to the desired load.

When you are testing connections/second, for example, look at the Details tab of your test. In this area you can see attempted connections and actual, established connections. If the number of attempted connections is equal to the number of connections you want to test, then the problem occurs in the establishment of the connections. This suggests that the limitation is in the device or system you're testing.

The rest of this section investigates parts of Avalanche's load generation that can produce inconsistencies. There are four main areas:

- Reporting
- Physical speed and capacity of the Load Generator
- Number of animating users
- Graphical representations

## **2.1 Reporting**

Avalanche reports its statistics every four seconds. The actual load and the desired load may not appear to match because of changes in the desired load over that reporting interval.

For rate-based load testing, the load generator examines the recent history to see how much load has been added; it divides this change in load by the amount of time over which that load was added. If this amount is less than the desired load, the Load Controller will issue an order to add more simulated users.

This approach works pretty well, but if you have defined a test that changes more often than every four seconds, the load graph that Avalanche produces may not look like the load graph you specified. Though Avalanche is flexible enough to handle these scenarios, there will be a visual incongruity.

## **2.2 Physical Speed and Capacity**

When the physical speed of the Load Generator is exhausted, it can no longer meet the desired load. There are limits on how much input/output can be generated based upon the capabilities of the Network Interface Card (NIC). Spirent has several different cards available to help you meet your requirements for testing large numbers of packets/second. Please contact a sales representative for more information.

In addition to the NIC, there are limits to CPU processor speeds. Some protocols like HTTPS, RTSP and RTP require significant processing for each action. Depending up the CPU, the number of actions that are started and already underway may be limited. This sort of performance limitation in the CPU is often indicated by an increase in the number of animated users. See Section 2.3 for more details.

The final physical piece is physical capacity. You can use the “resources” tab of Avalanche to see the amount of memory available (as well as the amount of memory and packet memory used).<sup>6</sup> If the amount of memory used is above 93% of the available, the Load Generator will not be able to add more users since there is insufficient memory to allocate to them. As in-process simulated users die, more memory is freed up. But rate-based metrics call for an ever increasing number of simulated users, so the few that die cannot free up enough memory for all of the new ones that the Load Generator wants to add. Load generation will continue at approximately the maximum that the appliance can generate—for better results, lower the amount of load you have specified and run the test again.

### 2.3 Animating Users

Simulated users that are actively traversing their Action List are called **animating users**. In rate-based metrics, the number of animating users may increase too quickly. Creating and maintaining users involves a fair number of processor cycles. The amount of time it takes to handle the work-in-progress influences (and restricts) the Load Generator’s ability to measure and add more load.

Transmission errors on the wire and the limitations of the device under testing can both create situations where users cannot process the entire Action List and so get stuck mid-way through their life-cycle, delayed by retransmissions and timeouts. These users do not die because they haven’t completed their work, but their existence hampers additional rate-based load.

To determine whether there is a problem with animating users, check the SimUsers bubble graph tab of the Statistics Details. If living and animating users account for more than 1% of the desired rate-based load, there is most likely a problem; you will also observe an ever increasing number of animating users. If this is happening in a rate-based test, the Load Generator will probably not be able to keep up. Reduce the amount of desired load and re-run the test.

### 2.4 Graphical Representation

When graphs are based upon statistics, it seems reasonable enough to conclude that they are based on facts. But statistics are slippery—as much art as they are science, their interpretations should be more suspect than they usually are.

One of the most common misuses of statistics is the “gee-whiz” graph, which inserts nonexistent drama into statistical trends. These can be purposefully constructed, but it’s easy to amplify trends accidentally, too. A huge discrepancy will look minor if the graph is zoomed out far enough. Likewise (and more relevant to our problem), a 1% error will rise precipitously as someone zooms in and may even look like a significant and imposing inconsistency.<sup>7</sup> The base for your graph can be something other than zero—check to make sure the traffic it describes is what you expect it to describe and that you are not too far zoomed in or out.

<sup>6</sup> Avalanche reports this information in real-time as tests are run. You can view physical capacity and other data by looking in the statistics details of a report that is underway.

<sup>7</sup> To read more about the implications of this, see *How to Lie with Statistics* by Darrell Huff.

### 3. Understanding the Problem Mathematically

Let's revisit Illustrations 1 and 2. In Illustration 1, you have set up your load specification to maintain 20,000 open connections. Maybe you know that your equipment can handle 20,000 connections, but you want to see if it can run at that level for 30 days. During this stress test, connections will open and close, but because the only task is to keep 20,000 connections open, the Load Controller simply tells the Load Engine that it should add a connection for each connection it removes. It's very clear-cut to maintain 20,000 connections. There is no impact from the time to look at traffic or time to generate it.

The first part of the concurrent metrics algorithm dictates how to generate load and continue the traffic already generated. This may be followed by a measure and report cycle, which in turn may be followed by a measure and add load cycle. The work-in-progress may have resulted in the death of many simulated users, so the load number reported may be temporarily below what is desired—however, the next cycle will begin with the addition of new users to get the load back up to the target amount.

The algorithm for rate-based load metrics is similar. The exception is that the Load Controller uses additional processor cycles to look at the history of measurements—this information overwhelmingly determines how much load should be added.

Back in Illustration 2, the system is trying to maintain a connection rate of 20,000 connections per second, while some number of connections, are closing. Since we know that it takes us time to look at the history and time to generate the traffic, we can add an adjustment factor that will offset these numbers. This is what the Load Controller does. The performance coefficient in the equation is a dynamic, well-educated approximation. The value will be close but not exact.

For example, illustration 4, shows this variation. At  $t=1.8$  we've added an additional 18,000 connections and we're on track. By  $t=1.9$ , we've added a further 1,000 connections. So far, so good. The Load Engine sends measurements back to the Load Controller and says we need another 1,000 connections over the next .1 seconds. It has taken time for this measurement to get to the Load Controller and the Load Controller knows this.

The Load Controller also knows that it will take a small amount of time to generate the traffic once it sends the command back to the Load Engine. The performance coefficient takes all of this into consideration and is part of the actual algorithm. With the performance coefficient in place, the algorithms can predict that since it's going to take more time than just .1 seconds, the Load Engine should add 1,200 connections. Every device and system under test is different. The performance coefficient is part of a complex algorithm to get this right, but it's possible that it could be slightly off. So by the time the Load Controller measures again at  $t=2.1$ , it is possible we could still be 100 connections below even though we added those 200 extra connections at  $t=1.9$ .<sup>8</sup>

<sup>8</sup> The Load Controller's algorithm will determine how many new connections to add in a current time interval based upon a) the requested number of connections per second, b) how long the tested system is taking to accept new connections, and c) the amount of time until the Load Controller next monitors connection progress.

This is a 0.5% error. If the scale of the x- and y-axes is large, this doesn't look like too much of a problem, but when you zoom in on the graph, this 0.5% error looks a lot bigger.

### Connections/Sec Rate Test — 20,000 Connections/Sec

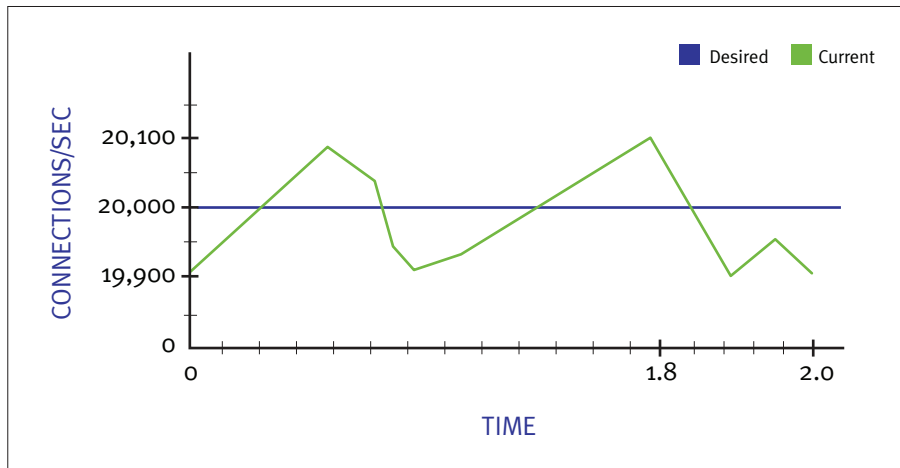


Illustration 4: Our Rate-based Test Zoomed in to t=1.8

Part of the problem here has to do with the fact that connections are discrete objects—there is no such thing as half a connection or half of a simulated user. The other part has to do with the fact that anything that measures a rate of change does so by approximation.

If we take any point on the curve of a rate of change, we get an instantaneous look at the rate of change—but this is always an approximation.<sup>9</sup> Even an instantaneous measurement requires some change in time and this change in time can have a huge impact when discrete math is used. So while calculus gives us the tools to tackle this problem, it also limits us from solving it completely.

The laws of calculus imply that the smoothness of the rate of change of time-based metrics (such as connection/sec or users/sec) is determined by the interval of the time being used in the calculation. The smoothest graphs and most accurate measurements are determined when the smallest possible time differences are used (i.e. as  $\Delta t$  approaches zero for the derivative value).

The timing of Avalanche's simulation is determined by discrete values such as connection establishment, latency and processing time; in real-world scenarios, these metrics can never be reduced down to zero. Because of this, the rate of change in the number of connections over time will never be as smooth as a theoretical model would allow, and changes to the number of users or connections can never be made instantaneously. Even if the time interval of calculation and response could be reduced down to near zero, calculus states that no change in the number of users or connections

<sup>9</sup> Rate-based metrics like connections/sec are **derivatives**.

is possible unless some amount of time has passed. The real-world conditions upon which Avalanche bases its testing mathematically limit how nimble it can be when making adjustments to these time-based values.

To return to the example of Mel's Diner, consider a graph charting restaurant capacity over time. A major component of the line for this graph is determined by the amount of time it takes for a typical diner to finish his meal. If everybody took 10 seconds to sit, eat, and leave, the restaurant could make very fast adjustments to the number of people being served at a particular time. But long dining time means more lag time in adjustment, and aberrant situations, such as that guy who takes 30 minutes to finish his dessert, introduce a variability that cannot be easily handled.

## 4. Tips and Tricks

There are still occasions when your actual rate may be more than just a small blip away from your desired rate. There are a number of methods that will reduce these more dramatic differences.

Let's say you've set up your test to add 250 connections/sec, but when you look at what is happening you see a large discrepancy between your desired rate and the actual one. The graph below starts at minute 3:10.

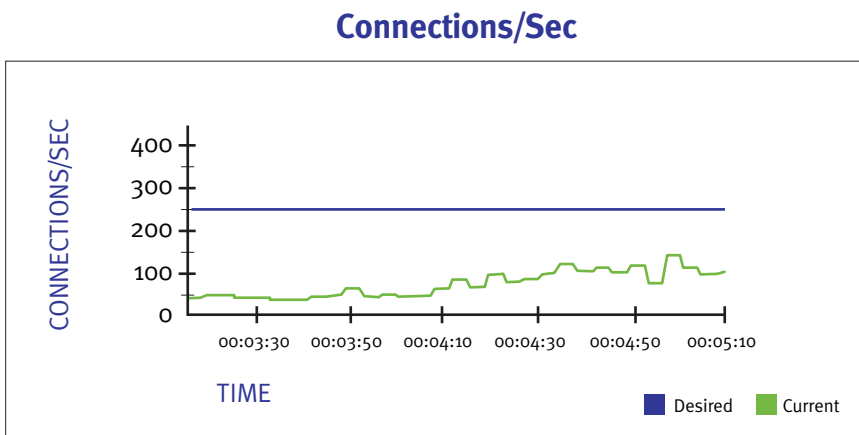


Illustration 5: A Discrepancy Between Desired Load and Actual Load

This section will take you through a series of troubleshooting techniques to get your actual load to correspond to your desired load.

### 4.1 Planning with Limits in Mind

The first thing to think about when you're planning a test is what your goal is—if you just want to test the bandwidth in a particular protocol, the simpler you can make the test, the better the information you're going to get about how the device is performing.

At other times, you would like to get bandwidth information, but also want a certain number of connections open and some new connections added each second. Under most circumstances, Avalanche can handle these complex testing requirements; however, there are physical limits. You shouldn't expect to test the upper limits of bandwidth with Avalanche if you're pulling a large object and opening an enormous number of connections/second at the same time. If you are encountering problems, it may be because your goals are too grand. Try testing capabilities separately. Customer support may be able to provide you with benchmarks for the particular hardware and software you are testing.

The general rule of thumb is that bandwidth equals the number of connections per second times the object size.

$bw = \text{connections/sec} * \text{object\_size}$

## **4.2 Areas to Troubleshoot**

Every test is composed of three main components—the User Profile, the Load Profile and the Action List. The following suggestions target these places.

### **4.2.1 User Profile**

The User Profile mimics a real user's behavior, so it can affect the amount of load that you can effectively test. Some user profile settings make each simulated user do more work, which means fewer resources are devoted to creating simulated users and more resources can be directed to testing connections or transactions.

The main influence is Think Time—the amount of time that the simulated users will spend in a given process (for example, how long they will take to browse one Web page before clicking through to another). Increasing the Think Time for concurrent tests allows you to test a higher load. From Avalanche's perspective, Think Time is really idle time for that simulated user, so it gives Avalanche time to do something else for other users. This time can be used to generate additional users or to add new connections.

For rate-based metrics, however, longer think times decrease performance. A longer Think Time means you need to create (and manage) more simulated users to achieve the desired rate—and each simulated user does less work. Increasing the Think Time for rate-based metrics, like connections/second or bandwidth, lengthens internal queues and makes more intensive demands on the processor.

If you are testing HTTP, also consider the impact of the persistence feature of HTTP 1.1. The persistence feature of HTTP 1.1 means that connections are shared. When your simulated users use HTTP 1.1, they will open one connection and be able to issue many requests through that single connection. The result is that you can test higher transactions per second by using HTTP 1.1.

### **4.2.2 Load Profile**

A load profile allows you to dictate how quickly to ramp up to your target rate and how to step the rate up thereafter. It is possible to describe a load profile with a duration that is too short to

actually carry out the amount of work specified in the workflow. Take, for example, Illustration 6. At first glance, there appears to be a problem—the test continues to generate load after 2 minutes, even though the desired load (blue) should be zero. The current load (green), continues to stair-step down the number of connections being tested.

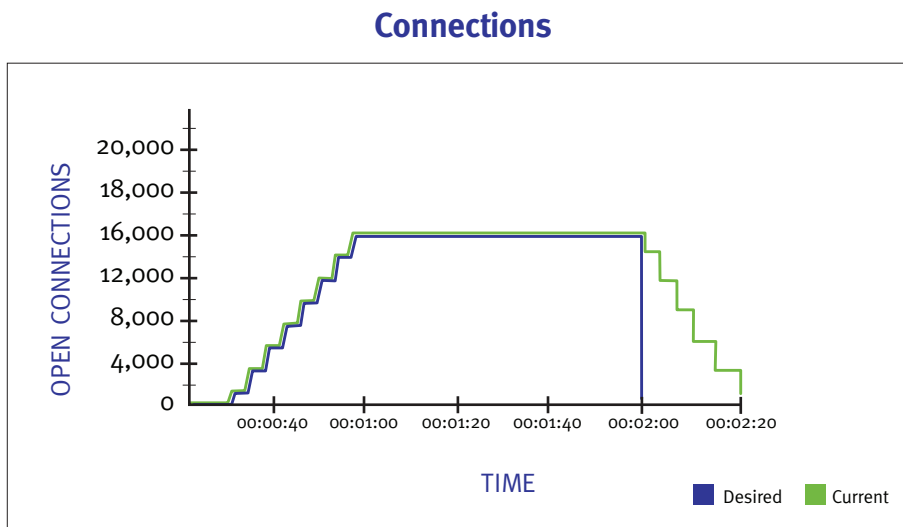


Illustration 6: Why Isn't the Load Decreasing?

It turns out that this is not actually a problem of the system, but one of workload. If the size of the objects being pulled is too big, Avalanche will keep the connections open as long as it needs to in order to finish—even if that goes past the time set in the duration portion of the load profile. After we decrease the size of the object (or lengthen the duration), the test will end as specified.

### 4.2.3 Action Lists

When you are testing bandwidth or connections/second, Spirent recommends that you place between four and 12 entries in your Action List (when you are testing a Web site, you can test however many entries are appropriate). If you have only one entry in your Action List, too many simulated users may be created—remember that after a simulated user travels to each URL, it dies and a new one is born: short lists mean too many simulated users.<sup>10</sup>

At the other extreme, too many entries on an Action List may reduce the efficiency with which the Load Engine creates new simulated users.<sup>11</sup> The entries in the Action List are really objects that get pulled across as simulated users encounter them. When taken with bandwidth and the number

<sup>10</sup> There is overhead associated with the creation of a simulated user's and for HTTP, keep-alive and non-keep-alive connections affect performance.

of connections, large objects can trigger the desired-actual discrepancy we've been addressing. In general, keep in mind the equation  $bandwidth = connections * object\_size$  when you are creating your tests. The upper limit for these variables is determined by the bandwidth capacity of the network and server devices being tested, among other things.

The final piece to watch with Action Lists is mixing protocols. If you are testing transactions or transactions/second, please note that your Action List should only involve HTTP and HTTPS. For other protocols, such as FTP, DNS, Streaming, POP3, SMTP, you should use the simulated user or simulated user/second specification—transactions are strictly for HTTP/HTTPS. The table below provides a summary of which tests can be run for which protocols.

	Simulated user-based	Connection-based	Transaction-based	Bandwidth	Body byte <sup>12</sup>
HTTP/HTTPS	X	X	X	X	X
FTP	X	X		X	
DNS	X			X	
Others	X	For all TCP-based protocols		X	

Table 1: Recommended Tests for Various Protocols

## 5. Conclusion

In this paper we've reviewed how load generation works and given some helpful hints on how to take advantage of Avalanche's rich set of customizable tests.

Spirent's algorithms give you an unprecedented amount of control. By allowing you to simulate interactions that are as demanding and volatile as what happens in the real world, Avalanche also produces truly meaningful test data—numbers you can count on. These capabilities carry with them some complexity, but Avalanche remains the most usable, reliable and robust testing tool available.

<sup>11</sup> This is because the Load Engine looks ahead to figure out how many simulated users to create; if the list is too long, the Load Engine may fail to add a sufficient number of simulated users at the right times. There are two issues here: first, it is harder to control rate-based loads that test transactions/second, and second, the Load Controller may get called less frequently, making its estimates less granular.

<sup>12</sup> Body byte metrics should be run with rate-based constraints.