



## jetNEXUS Enterprise Traffic Manager

### TRAFFIC VALUATION AND PRIORITIZATION WITH ENTERPRISE TRAFFIC MANAGER

#### **Application Delivery Control & Traffic Management**

The jetNEXUS Enterprise Traffic Manager is a powerful, fully fault tolerant load balancing solution with advanced traffic management and routing capabilities, making applications faster, more reliable, secure and easier to manage.

The jetNEXUS Enterprise Traffic Manager offers massive performance and the most comprehensive Application Delivery Control feature set available on the market, including advanced traffic scripting language and JAVA extensions and interoperability.

# Contents

<b>Introduction</b> .....	<b>3</b>
An example .....	3
<b>Designing a service policy</b> .....	<b>4</b>
Enterprise Traffic Manager TrafficScript Rules.....	5
<b>Service Level Monitoring</b> .....	<b>6</b>
Example: Simple Differentiation.....	7
<b>Application Traffic Inspection</b> .....	<b>8</b>
Remembering the Classification with a Cookie .....	8
<b>Request Rate Shaping</b> .....	<b>10</b>
Using a Rate Class.....	10
Rate Classes with Keys.....	11
Applying service policies with rate shaping.....	12
<b>Bandwidth Shaping</b> .....	<b>12</b>
Using Bandwidth Shaping .....	13
Example: Managing Flash Floods .....	13
<b>Traffic Routing and Termination</b> .....	<b>13</b>
Example: Ticket Booking Systems.....	14
Example: Prioritizing Resource Usage .....	15
<b>Selective Traffic Optimization</b> .....	<b>16</b>
Content Compression.....	16
Content Caching.....	16
<b>Custom Logging</b> .....	<b>17</b>
<b>Conclusion</b> .....	<b>18</b>

## Introduction

Whether you are running an eCommerce web site, online corporate services or an internal intranet, there's always the need to squeeze more performance from limited resources and to ensure that your most valuable users get the best possible levels of service from the services you are hosting.

Flash crowd events, greedy or malicious users and poorly-performing applications – these all affect the level of service you deliver. The powerful traffic management capabilities of jetNEXUS Traffic Manager let you define the service policies that deal with these problems, and apply them easily and non-intrusively to all your traffic.

### An example

Imagine that you are running a successful gaming service in a glamorous location. The usage of your service is growing daily, and many of your long-term users are becoming very valuable.

Unfortunately, much of your bandwidth and server hits are taken up by competitors' robots that screen-scrape your betting statistics, and poorly-written bots that spam your gaming tables and occasionally place low-value bets.

At certain times of the day, this activity is so great that it impacts the quality of the service you deliver, and your most valuable customers are affected.

With Enterprise Traffic Manager's ability to measure, classify and prioritize traffic, you can construct a service policy that comes into effect when your web site begins to run slowly.

The policy can inspect and classify each request, looking at information like request URLs, cookies and form data, and it can recording data in cookies.

Having classified the request, the policy then enforces different levels of service:

- Competitor's screen-scraping robots are tightly restricted to one request per second each. A ten-second delay reduces the value of the information they screen-scrape.
- Users who have not yet logged in are limited to a small proportion of your available bandwidth and directed to a pair of basic web servers, thus reserving capacity for users who are logged in.
- Users who have made large transactions in the past are tagged with a cookie and the performance they receive is measured. If they are receiving poor levels of service (over 100ms response time), then some of the transaction servers are reserved for these high-value users and the activity of other users is shaped by a system-wide queue.

Whether you are operating a gaming service, a content portal, a B2B or B2C eCommerce site or an internal intranet, this kind of service policy can help ensure that key customers get the best possible service, minimise the churn of valuable users and prevent undesirable visitors from harming the service to the detriment of others.



## Designing a service policy

This white paper describes techniques to address the following problems:

“I want to guarantee certain levels of service for certain users.”

“I want to prioritize some transactions over others.”

“I want to restrict the activities of certain users.”

To address these problems, an administrator must consider the following questions:

1. Under what circumstances do you want the policy to take effect?
2. How do you wish to categorise your users?
3. How do you wish to apply the differentiation?

One or more TrafficScript rules can be used to apply the policy. They take advantage of the following jetNEXUS Enterprise Traffic Manager features:

### 1. When does the policy take effect?

- **Service Level Monitoring** – Measure system performance, and apply policies only when they are needed.
- **Custom Logging** – Log and analyse activity to record and validate policy decisions.

### 2. How are users categorized?

- **Application traffic inspection** – Determine source, user, content, value; XML processing with XPath searches and calculations.

### 3. How are they given different levels of service?

- **Request Rate Shaping** – Apply fine-grained rate limits for transactions.
- **Bandwidth Control** – Allocate and reserve bandwidth.
- **Traffic Routing and Termination** – Route high and low priority traffic differently; Terminate undesired requests.
- **Selective Traffic Optimisation** – Selective caching and compression.

## jetNEXUS Enterprise Traffic Manager TrafficScript Rules

The jetNEXUS ETM functions as an application proxy, accepting clients' requests and load-balancing them across a number of back-end servers.

Central to Enterprise Traffic Manager is the concept of TrafficScript rules. Using the simple TrafficScript programming language, you can write scripts that:

- Inspect all aspects of requests and responses, and perform complex calculations such as XSLT transforms.
- Modify request or response data as it passes through the Enterprise Traffic Manager.
- Fine-tune the behaviour of Enterprise Traffic Manager for each request.

TrafficScript includes a large number of helper functions to facilitate the inspection, transformation and routing of traffic.

These rules are run at two different times:

- Whenever Enterprise Traffic Manager receives a client request (a request rule);
- Whenever Enterprise Traffic Manager receives a response from the back-end server (a response rule).

For example, the following TrafficScript request rule inspects HTTP requests. If the request is for a .jsp page, the rule looks at the client's 'Priority' cookie and routes the request to the 'high-priority' or 'low-priority' server pools as appropriate:

```
$url = http.getPath();
if( string.endsWith( $url, ".jsp" ) ) {
    $cookie = http.getCookie( "Priority" );

    if( $cookie == "high" ) {
        pool.use( "high-priority" );
    } else {
        pool.use( "low-priority" );
    }
}
```

Generally, if you can describe the traffic management logic that you require, it is possible to implement it using TrafficScript.

This document will refer to TrafficScript rules extensively as it describes how to implement service policies. If you would like more information on TrafficScript, please refer to the jetNEXUS Enterprise Traffic Manager page here: <http://www.jetnexus.com/enterprise-traffic-manager.html>.

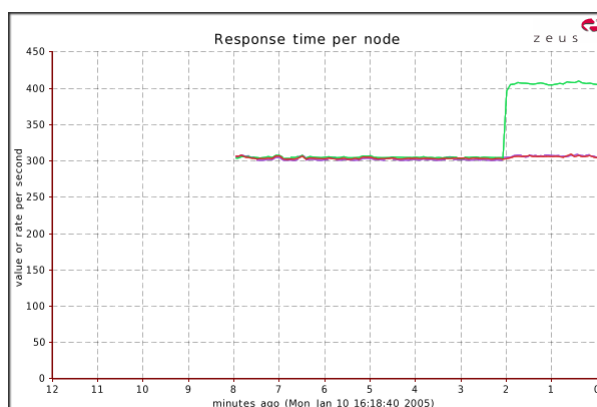
## Service Level Monitoring

Using Service Level Monitoring, jetNEXUS ETM can measure and react to changes in response times for your hosted services, by comparing response times to a desired time.

You configure Service Level Monitoring by creating a Service Level Monitoring Class (SLM Class). The SLM Class is configured with the desired response time (for example, 100ms), and some thresholds that define actions to take<sup>1</sup>.

```
$url = http.getPath();  
if( string.startsWith( $url, "/servlet/" ) {  
    connection.setServiceLevelClass( "Java servlets" );  
}
```

You can then monitor the performance figures generated by the 'Java servlets' SLM class to discover the response times, and the proportion of requests that fall outside the desired response time:



Once requests are monitored by an SLM Class, you can discover the proportion of requests that meet the desired response time within a TrafficScript rule. This makes it possible to implement TrafficScript logic that is only called when services are underperforming.

<sup>1</sup> For example, if fewer than 80% of requests meet the desired response time, ETM can log a warning; if fewer than 50% meet the desired time, ETM can raise a system alert.

### Example: Simple Differentiation

Suppose we had a TrafficScript rule that tested to see if a request came from a 'high value' customer.

- When our service is running slowly, high-value customers should be sent to one server pool ('gold') and other customers sent to a lower-performing server pool ('bronze').
- However, when the service is running at normal speed, we want to send all customers to all servers (the server pool named 'all servers').

The following TrafficScript rule describes how this logic can be implemented:

```
# Monitor all traffic with the 'response time' SLM class, which is
# configured with a desired response time of 200ms

connection.setServiceLevelClass( "response time" );

# Now, check the historical activity (last 10 seconds) to see if it's
# been acceptable (more than 90% requests served within 200ms)

if( slm.conforming( "response time" ) > 90 ) {
  # select the 'all servers' server pool and terminate the rule
  pool.use( "all servers" );
}

# If we get here, things are running slowly

# Here, we decide a customer is 'high value' if they have a login cookie,
# so we penalize customers who are not logged in. You can put your own
# test here instead

$logincookie = http.getCookie( "Login" );
if( $logincookie ) {
  pool.use( "gold" );
} else {
  pool.use( "bronze" );
}
```

## Application Traffic Inspection

There's no limit to how you can inspect and value your traffic. ETM's Application Traffic Inspection lets you look at any aspect of a client's request, so that you can then categorize them as you need.

For example:

```
# What is the client asking for?
$url = http.getPath();

# ... and the QueryString
$qqs = http.getQueryString();

# Where has the client come from?
$referrer = http.getHeader( "Referer" );

# What sort of browser is the client using?
$ua = http.getHeader( "User-Agent" );

# Is the client trying to spend more than $49.99?
if( http.getPath() == "/checkout.cgi" &&
    http.getFormParam( "total" ) > 4999 ) {
    ...
}

# What's the value of the CustomerName field in the XML purchase order
# in the SOAP request?
$body = http.getBody();
$name = xml.xpath.matchNodeSet( $body, "", "//Info/CustomerName/text()" );

# Take the name, post it to a database server with a web interface and
# inspect the response. Does the response contain the value 'Premium'?
$response = http.request.post( "http://my.database.server/query",
    "name=".string.htmlEncode( $name ) );
if( string.Contains( $response, "Premium" ) ) {
    ...
}
```

### Remembering the Classification with a Cookie

Often, it only takes one request to identify the status of a user, but you want to remember this decision for all subsequent requests. For example, if a user places an item in his shopping cart by accessing the URL '/cart.php', then you want to remember this information for all of his subsequent requests.

Adding a response cookie is the way to do this. You can do this in a Response Rule with the 'http.setResponseCookie()' function:

```
if( http.getPath() == "/cart.php" ) {
  http.setResponseCookie( "GotItems", "Yes" );
}
```

You can do it in a Request Rule as follows:

```
if( http.getPath() == "/cart.php" ) {
  http.setHeader( "Set-Cookie", "GotItems=Yes" );
}
```

This cookie will be sent by the client on every subsequent request, so to test if the user has placed items in his shopping cart, you just need to test for the presence of the 'GotItems' cookie in each request rule:

```
if( http.getCookie( "GotItems" ) ) {
  ...
}
```

If necessary, you can encrypt and sign the cookie so that it cannot be spoofed or reused:

```
# Setting the cookie

# Create an encryption key using the client's IP address and user agent
# Encrypt the current time using encryption key; it can only be decrypted
# using the same key

$key = http.getHeader( "User-Agent" ) . ":" . request.getRemoteIP();
$encrypted = string.encrypt( sys.time(), $key );
$encoded = string.hexencode( $encrypted );
http.setHeader( "Set-Cookie", "GotItems=" . $encoded );

# Validating the cookie
$isValid = 0;
if( $cookie = http.getCookie( "GotItems" ) ) {

  $encrypted = string.hexdecode( $cookie );
  $key = http.getHeader( "User-Agent" ) . ":" . request.getRemoteIP();
  $secret = string.decrypt( $encrypted, $key );

  # If the cookie has been tampered with, or the ip address or user
  # agent differ, the string.decrypt will return an empty string.
  # If it worked and the data was less than 1 hour old, it's valid:
  if( $secret && sys.time()-$secret < 3600 ) {
    $isValid = 1;
  }
}
```

## Request Rate Shaping

Having decided when to apply your service policy (using Service Level Monitoring), and classified your users (using Application Traffic Inspection), you now need to decide how to prioritize valuable users and penalize undesirable ones.

ETM's Request Rate Shaping capability is used to apply maximum request rates:

- On a global basis (“no more than 100 requests per second to my application servers”);
- On a very fine-grained per-user or per-class basis (“no user can make more than 10 requests per minute to any of my statistics pages”).

You can construct a service policy that places limits on a wide range of events, with very fine grained control over how events are identified. You can impose per-second and per-minute rates on these events.

For example:

- You can rate-shape individual web spiders, to stop them overwhelming your web site. Each web spider, from each remote IP address, can be given maximum request rates.
- You can throttle individual SMTP connections, or groups of connections from the same client, so that each connection is limited to a maximum number of sent emails per minute.
- You may also rate-shape new SMTP connections, so that a remote client can only establish new connections at a particular rate.
- You can apply a global rate shape to the number of connections per second that are forwarded to an application.
- You can identify individual user's attempts to log in to a service, and then impede any dictionary-based login attacks by restricting each user to a limited number of attempts per minute.

Request Rate Limits are imposed using the TrafficScript `rate.use()` function, and you can configure per-second and per-minute limits in the rate class. Both limits are applied (note that if the per-minute limit is more than 60-times the per-second limit, it has no effect).

### Using a Rate Class

Rate classes function as queues. When the TrafficScript `rate.use()` function is called, the connection is suspended and added to the queue that the rate class manages. Connections are then released from the queue according to the per-second and per-minute limits.

There is no limit to the size of the backlog of queued connections. For example, if 1000 requests arrived in quick succession to a rate class that admitted 10 per second, 990 of them would be immediately queued. Each second, 10 more requests would be released from the front of the queue.

While they are queued, connections may time out or be closed by the remote client. If this happens, they are immediately discarded.

You can use the `rate.getBacklog()` function to discover how many requests are currently queued. If the backlog is too large, you may decide to return an error page to the user rather than risk their connection timing out. For example, to rate shape jsp requests, but defer requests when the backlog gets too large:

```
$url = http.getPath();

if( string.endsWith( $url, ".jsp" ) ) {
    if( rate.getBacklog( "shape requests" ) > 100 ) {
        http.redirect( "http://mysite/too_busy.html" );
    } else {
        rate.use( "shape requests" );
    }
}
```

### Rate Classes with Keys

In many circumstances, you may need to apply more fine-grained rate-shape limits. For example, imagine a login page; we wish to limit how frequently each individual user can attempt to log in to just 2 attempts per minute.

The `rate.use()` function can take an optional 'key' which identifies a specific instance of the rate class. This key can be used to create multiple, independent rate classes that share the same limits, but enforce them independently for each individual key.

For example, the 'login limit' class is restricted to 2 requests per minute, to limit how often each user can attempt to log in:

```
$url = http.getPath();
if( string.endsWith( $url, "login.cgi" ) ) {
    $user = http.getFormParam( "username" );
    rate.use( "login limit", $user );
}
```

This rule can help to defeat dictionary attacks where attackers try to brute-force crack a user's password. The rate shaping limits are applied independently to each different value of `$user`. As each new user accesses the system, they are limited to 2 requests per minute, independently of all other users who share the "login limit" rate shaping class.

## Applying service policies with rate shaping

Of course, once you've classified your users, you can apply different rate settings to different categories of users:

```
# If they have an odd-looking user agent, or if there's no host header,  
# the client is probably a web spider. Limit it to 1 request per second.  
$ua = http.getHeader( "User-Agent" );  
  
if( ! string.startsWith( $ua, "Mozilla/" ) &&  
    ! string.startsWith( $ua, "Opera/" ) ||  
    ! http.getHeader( "Host" ) ) {  
    rate.use( "spiders", request.getRemoteIP() );  
}
```

If the service is running slowly, rate-shape users who have not placed items into their shopping cart with a global limit, and rate-shape other users to 8 requests per second each:

```
if( slm.conforming( "timer" ) < 80 ) {  
    $cookie = request.getCookie( "Cart" );  
    if( ! $cookie ) {  
        rate.use( "casual users" );  
    } else {  
        # Get a unique id for the user  
        $cookie = request.getCookie( "JSPSESSIONID" );  
        rate.use( "8 per second", $cookie );  
    }  
}
```

## Bandwidth Shaping

Bandwidth shaping allows ETM to limit the number of bytes per second used by inbound or outbound traffic, for an entire service, or by the type of request.

Bandwidth limits are automatically shared and enforced across all the ETM machines in a cluster. Individual ETM machines take different proportions of the total limit, depending on the load on each, and unused bandwidth is equitably allocated across the cluster depending on the need of each machine.

Like Request Rate Shaping, you can use Bandwidth shaping to limit the activities of subsets of your users. For example, you may have a 20Mbits/s network connection which is being over-utilised by a certain type of client, which is affecting the responsiveness of the service. You may therefore wish to limit the bandwidth available to those clients to 2Mbits/s.

## Using Bandwidth Shaping

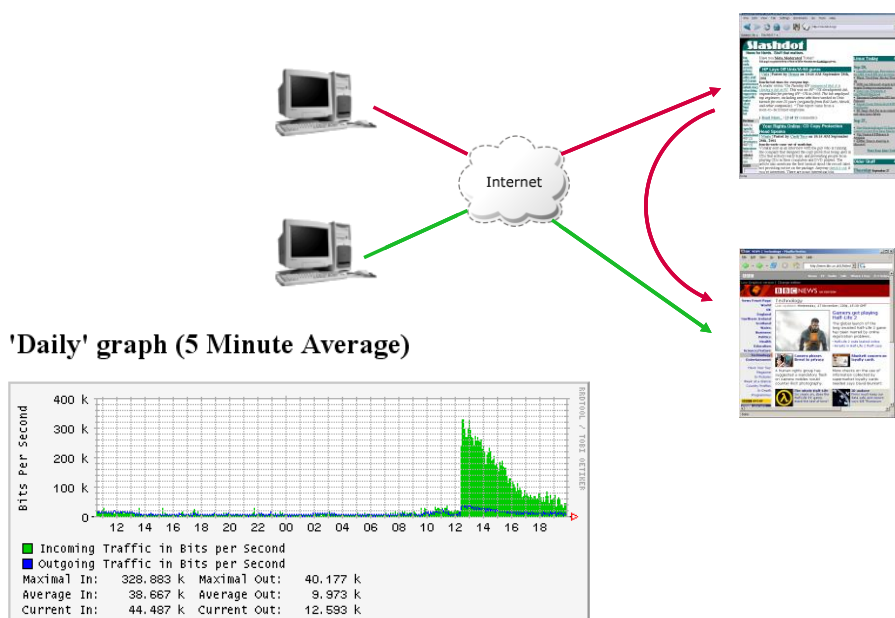
Like Request Rate Shaping, you configure a Bandwidth class with a maximum bandwidth limit. Connections are allocated to a class as follows:

```
response.setBandwidthClass( "class name" );
```

All of the connections allocated to the class share the same bandwidth limit.

### Example: Managing Flash Floods

The following example helps to mitigate the ‘Slashdot Effect’, a common example of a Flash Flood problem. In this situation, a web site is overwhelmed by traffic as a result of a high-profile link (for example, from the Slashdot news site), and the level of service that regular users experience suffers as a result.



The example looks at the ‘Referer’ header, which identifies where a user has come from to access a web site. If the user has come from ‘slashdot.org’, he is tagged with a cookie so that all of his subsequent requests can be identified, and he is allocated to a low-bandwidth class:

```
$referrer = http.getHeader( "Referer" );
if( string.contains( $referrer, "slashdot.org" ) ) {
    http.addResponseHeader(
        "Set-Cookie", "slashdot=1" );
    connection.setBandwidthClass( "slashdot" );
}

if( http.getCookie( "slashdot" ) ) {
    connection.setBandwidthClass( "slashdot" );
}
```

## Traffic Routing and Termination

Different levels of service can be provided by different traffic routing, or in extreme events, by dropping some requests.

For example, some large media sites provide different levels of content; high-bandwidth rich media versions of news stories are served during normal usage, and low-bandwidth versions which are served when traffic levels are extremely high. Many websites provide flash-enabled and simple HTML versions of their home page and navigation.

This is also commonplace when presenting content to a range of browsing devices with different capabilities and bandwidth.

The switch between high and low bandwidth versions could take place as part of a service policy: as the service begins to under-perform, some (or all) users could be forced onto the low-bandwidth versions so that a better level of service is maintained.

```
# Forcibly change requests that begin /high/ to /low/  
$url = http.getPath();  
if( string.startsWith( $url, "/high" ) ) {  
    $url = string.replace( $url, "/high", "/low" );  
    http.setPath( $low );  
}
```

### Example: Ticket Booking Systems

Ticket Booking systems for major events often suffer enormous floods of demand when tickets become available.

You can use ETM's request rate shaping system to limit how many visitors are admitted to the service, and if the service becomes overwhelmed, you can send back a 'please try again' message rather than keeping the user 'on hold' in the queue indefinitely.

Suppose the 'booking' rate shaping class is configured to admit 10 users per second, and that users enter the booking process by accessing the URL "/bookevent?eventID=<id>". This rule ensures that no user is queued for more than 30 seconds, by keeping the queue length to no more than 300 users (10 users/second \* 30 seconds):

```
# limit how users can book events  
  
$url = http.getPath();  
if( $url == "/bookevent" ) {  
    # How many users are already queued?  
    if( rate.getBacklog( "booking" ) > 300 ) {  
        http.redirect( "http://www.mysite.com/too_busy.html");  
    } else {  
        rate.use( "booking" );  
    }  
}
```

### Example: Prioritizing Resource Usage

In many cases, the resources are limited and when a site is overwhelmed, users' requests still need to be served.

Consider the following scenario:

- The site runs a cluster of 4 identical application servers ('servers '1' to '4');
- Users are categorized into casual visitors and customers; customers have a 'Cart' cookie, and casual visitors do not.

Our goal is to give all users the best possible level of service, but if customers begin to get a poor level of service, we want to prioritize them over casual visitors. We desire that more than 80% of customers get responses within 100ms.

This can be achieved by splitting the 4 servers into 2 pools: the 'allservers' pool contains servers 1 to 4, and the 'someservers' pool contains servers 1 and 2 only.

When the service is poor for the customers, we will restrict the casual visitors to just the 'someservers' pool. This effectively reserves the additional servers 3 and 4 for the customers' exclusive use.

The following code uses the 'response' SLM class to measure the level of service that customers receive:

```
$customer = http.getCookie( "Cart" );  
if( $customer ) {  
    connection.setServiceLevelClass( "response" );  
    pool.use( "allservers" );  
} else {  
    if( slm.conforming( "response" ) < 80 ) {  
        pool.use( "someservers" );  
    } else {  
        pool.use( "allservers" );  
    }  
}
```

## Selective Traffic Optimization

Some of the features in ETM can be used to improve the end user's experience, but they take up resources on the ETM system:

- **Content Compression** reduces the bandwidth used in responses and gives better response times, but it takes considerable CPU resources and can degrade performance.
- **Content Caching** can give much faster responses, and it is possible to cache multiple versions of content for each user. However, this consumes memory on the ETM system.

Both of these features can be enabled and disabled on a per-user basis, as part of a service policy.

### Content Compression

Use the `http.compress.enable()` and `http.compress.disable()` TrafficScript functions to control whether or not ETM will compress response content to the remote client.

Note that ETM will only compress content if the remote browser has indicated that it supports compression.

On a lightly loaded system, it's appropriate to compress all response content whenever possible<sup>2</sup>:

```
http.compress.enable();
```

On a ETM where the CPU usage is becoming too high, you can selectively compress content:

```
# Don't compress by default
http.compress.disable();

if( $isvaluable ) {
  # do compress in this case
  http.compress.enable();
}
```

### Content Caching

ETM can cache multiple different versions of a HTTP response. For example, if your home page is generated by an application that customizes it for each user, ETM can cache each version separately, and return the correct version from the cache for each user who accesses the page.

---

<sup>2</sup> You can also enable content compression for all applicable responses by enabling the Virtual Server 'Content Compression' setting.

ETM's cache has a limited size so that it does not consume too much memory and cause performance to degrade. You may wish to prioritize which pages you put in the cache, using the `http.cache.disable()` and `http.cache.enable()` TrafficScript functions.

Note: you also need to enable Content Caching in your ETM Virtual Server configuration; otherwise the TrafficScript cache control functions will have no effect.

```
# Get the user name
$user = http.getCookie( "UserName" );

# Don't cache any pages by default:
http.cache.disable();

if( $isvaluable ) {
  # Do cache these pages for better performance.
  # Each user gets a different version of the page, so we need to cache
  # the page indexed by the user name.
  http.cache.setkey( $user );
  http.cache.enable();
}
```

## Custom Logging

A service policy can be complicated to construct and implement.

The TrafficScript functions `log.info()`, `log.warn()` and `log.error()` are used to write messages to the ETM event log, and so are very useful debugging tools to assist in developing complex TrafficScript rules.

For example, the following code:

```
if( $isvaluable && slm.conforming( "timer" ) < 70 ) {
  log.info( "User ".$user." needs priority" );
}
```

... will append the following message to your error log file:

```
$ tail ZEUSHOME/zxtm/log/errors
[20/Jan/2004:10:24:46 +0000] INFO:VSName:Rule=RuleName:User Jack needs priority
```

You can also inspect your error log file by viewing the 'Event Log' on the ETM Admin Server.

When you are debugging a rule, you can use `log.info()` to print out progress messages as the rule executes. The `log.info()` function takes a string parameter; you can construct complex strings by appending variables and literals together using the `'.'` operator:

```
$msg = "Received ".connection.getDataLen()." bytes.";
log.info( $msg );
```

The functions `log.warn()` and `log.error()` are similar to `log.info()`. They prefix error log messages with a higher priority - either "WARN" or "ERROR". The Event Log viewer can be used to filter out low-priority messages.

You should be careful when printing out connection data verbatim, because the connection data may contain control characters or other non-printable characters. You can encode data using either `'string.hexEncode()'` or `'string.escape()'`.

You should use `'string.hexEncode()'` if the data is binary, and `'string.escape()'` if the data contains readable text with a small number of non-printable characters.

## Conclusion

ETM is a powerful toolkit for network and application administrators. This white paper described a number of techniques to use tools in the kit to solve a range of traffic valuation and prioritization tasks.

For more information about the Enterprise Traffic Manager and TrafficScript can be used to manipulate your application traffic to improve end-user experience, feel free to get in touch with us on: [info@jetnexus.com](mailto:info@jetnexus.com).

## Copyright

© jetNEXUS Ltd 2011. Copyright in this document belongs to jetNEXUS Ltd. All rights are reserved.

Our web site contains a wealth of information on our products, services and solutions, as well as customer case studies and press information. For more information, please visit:

[www.jetNEXUS.com](http://www.jetNEXUS.com)



**jetNEXUS**

**Tel:** +44 (0) 870 382 5050

**Fax:** +44 (0) 870 382  
5520

**Email:** [info@jetnexus.com](mailto:info@jetnexus.com)

**Web:** [www.jetnexus.com](http://www.jetnexus.com)

jetNEXUS is an international supplier of Load Balancing, Application Acceleration and Application Delivery technology. Our product portfolio is accessible to a wide and varied client base, ranging from simple, cost effectively focused solutions to Enterprise-grade application delivery controllers. We understand that this technology is mission critical and as such, maintain a tenacious focus on the quality of our products and technical support.