

BIG-IP® Acceleration: Concepts

Version 12.1.0



Table of Contents

Introducing Acceleration.....	11
Overview: Introduction to acceleration.....	11
Origin web server load balancing.....	11
About data centers.....	12
Data compression.....	12
Data deduplication.....	12
Optimization of TCP connections.....	12
Caching objects.....	13
Optimization of HTTP protocol and web applications.....	13
Overview: BIG-IP Acceleration.....	14
Application management.....	14
Application monitoring.....	14
Deployment of Distributed BIG-IP Application Acceleration.....	14
Management of requests to origin web servers.....	15
Management of responses to clients.....	16
Flow of requests and responses.....	16
About symmetric optimization using iSession on BIG-IP systems.....	17
 Accelerating Traffic with Acceleration Profiles.....	 19
About HTTP compression profiles.....	19
HTTP Compression profile options.....	19
About Web Acceleration profiles.....	19
Web Acceleration profile settings.....	20
Meta characters.....	21
Web Acceleration Profile statistics description.....	23
About iSession profiles.....	24
Screen capture showing compression settings.....	24
About CIFS traffic optimization.....	25
About MAPI optimization.....	25
About TCP profiles.....	26
About tcp-lan-optimized profile settings.....	26
About tcp-mobile-optimized profile settings.....	26
About mptcp-mobile-optimized profile settings.....	27
About tcp-wan-optimized profile settings.....	28
About HTTP2 profiles.....	28
About HTTP/2 profiles.....	29
HTTP/2 profile settings.....	30
About SPDY profiles.....	31
About NTLM profiles.....	32

About OneConnect profiles.....	33
OneConnect and HTTP profiles.....	34
OneConnect and SNATs.....	34
OneConnect and NTLM profiles.....	35
Managing Traffic with Bandwidth Controllers.....	37
Overview: Bandwidth control management.....	37
Bandwidth controllers vs. rate shaping.....	37
About static bandwidth control policies.....	37
About dynamic bandwidth control policies.....	38
Example of a dynamic bandwidth control policy.....	39
Managing Traffic with Rate Shaping.....	41
Introduction to rate shaping.....	41
About rate classes.....	41
Rate class name.....	42
Base rate.....	42
Ceiling rate.....	42
Burst size.....	42
Depleting the burst reservoir.....	43
Replenishing a burst reservoir.....	43
About specifying a non-zero burst size	43
About the direction setting.....	44
About the parent class.....	45
About shaping policy.....	45
About queue method.....	45
About drop policy.....	46
Using Acceleration Policies to Manage and Respond to HTTP Requests.....	47
Overview: Acceleration policies.....	47
Policies screen access.....	47
Types of acceleration policies.....	48
BIG-IP acceleration policies options.....	48
Acceleration policy selection.....	48
Customization of acceleration policies.....	49
Creation of user-defined policies.....	49
Publication of acceleration policies.....	49
About the Acceleration Policy Editor role.....	50
Acceleration policies exported to XML files.....	50
Overview: Policy Matching.....	50
Resolution rules when multiple nodes match.....	50
Unmatched requests.....	52
An example matching rule.....	53
Overview: Policy Editor screen.....	53

Policy Editor screen parts.....	54
Policy Tree.....	55
Acceleration policy rule inheritance.....	55
Inheritance rule parameters.....	56
Inheritance rule parameters override.....	57
Policy Tree modification for an acceleration policy.....	59
Overview: HTTP header parameters.....	59
Requirements for servicing requests.....	59
About the HTTP request process.....	60
Requirements for caching responses.....	61
About the HTTP responses process.....	61
Configuration of rules based on HTTP request headers.....	61
Configuration of rules based on HTTP response headers.....	66
Regular expressions and meta tags for rules.....	68
Management of Cache-Control response headers.....	68
X-WA-Info response headers.....	69
Reference summary for HTTP data.....	72
HTTP request data type parameters.....	72
Response status codes.....	72
S code definitions.....	73
HTTP data types for regular expression strings.....	74
Max age value for compiled responses.....	76
Meta characters.....	76
Advanced Debug settings for General Options	78
Differentiating Requests and Responses with Variation Rules.....	79
Overview: Variation rules.....	79
Cache efficiency improvement.....	79
User-specific content.....	80
Definition of variation rules parameters.....	80
Value groups.....	80
Management of conflicting rules parameters.....	81
Managing Compiled Responses with Assembly Rules.....	83
Overview: Assembly rules.....	83
Management of content served from origin web servers.....	83
Proxying Requests and Responses.....	85
Overview: Proxying rules.....	85
Proxying rules parameters.....	85
Managing Requests and Responses with Lifetime Rules.....	87
Overview: Lifetime rules.....	87

Lifetime managed requests.....	87
Lifetime managed responses.....	87
About specifying the amount of time to store cached content.....	88
About serving cached content when origin web server content is unavailable.....	89
About preserving origin web server headers and directives to downstream devices.....	89
Custom Cache-Control directives.....	89
About replacing origin web server headers and directives with a no-cache directive.....	90
Invalidating Cached Content.....	91
Overview: Invalidating cached content for an application.....	91
Overview: Invalidating cached content for a node.....	91
Invalidations triggers.....	91
Invalidations lifetime.....	92
Invalidations rules parameters.....	93
Request header matching criteria.....	93
Cached content to invalidate.....	93
Managing Object Types.....	95
Overview: Object classification.....	95
Classification by object type.....	95
Classification by group.....	95
Management of object types.....	95
Caching Objects in a VIPRION Cluster.....	97
Overview: Acceleration in a cluster.....	97
Immediately Caching Dynamic Objects.....	99
Overview: Caching an object on first hit.....	99
Accelerating Parallel HTTP Requests.....	101
Overview: HTTP request queuing.....	101
Managing HTTP Traffic with the HTTP/2 Profile.....	103
Overview: Managing HTTP Traffic with the HTTP/2 profile.....	103
About HTTP/2 profiles.....	103
HTTP/2 profile settings.....	104
Managing HTTP Traffic with the SPDY Profile.....	107
Overview: Managing HTTP traffic with the SPDY profile.....	107

SPDY profile settings.....	108
Accelerating Requests and Responses with Intelligent Browser Referencing.....	111
Overview: Reducing conditional GET requests with Intelligent Browser Referencing....	111
About conditional GET requests.....	111
About Intelligent Browser Referencing for HTML.....	111
About Intelligent Browser Referencing for cascading style sheet files.....	112
About the adaptive Intelligent Browser Referencing lifetime.....	112
Intelligent Browser Referencing example.....	113
Advanced IBR settings for general options	114
Accelerating JavaScript and Cascading Style Sheet Files.....	115
Overview: Accelerating JavaScript, HTML, cascading style sheet, and inline image files.....	115
About minification of JavaScript, HTML, and cascading style sheet content.....	115
About reordering cascading style sheet and JavaScript URLs and content.....	115
About inlining documents and image data.....	116
About concatenation of JavaScript and cascading style sheet files.....	116
About DNS prefetching.....	116
Establishing Additional TCP Connections with MultiConnect.....	119
Overview: Accelerating requests and responses with MultiConnect.....	119
Optimization of TCP connections.....	119
MultiConnect example.....	120
Serving Specific Hyperlinked Content with Parameter Value Substitution.....	121
Overview: Serving specific hyperlinked content with parameter value substitution.....	121
About configuring value substitution parameters for an assembly rule.....	121
About using number randomizer for parameter value substitution.....	122
A parameter value substitution example.....	122
Accelerating Access to PDF Content.....	125
Overview: Accelerating access to PDF content with PDF linearization.....	125
Accelerating Images with Image Optimization.....	127
Overview: Accelerating images with image optimization.....	127
Optimization of image format.....	127
Optimization with JPEG-XR.....	127
Optimization with WebP.....	128
Optimization with file compression.....	128
Optimization of headers.....	128
Optimization of sampling factor.....	129
Optimization with progressive encoding.....	129

Optimization of color values.....	129
Accelerating Video Streams with Video Delivery Optimization.....	131
About video delivery optimization.....	131
About caching video segments by location.....	131
About caching popular content.....	131
About video delivery optimization cache priority.....	131
About globally configuring video delivery optimization.....	131
About video delivery optimization bit rate selection.....	131
About the video Quality of Experience profile.....	132
About mean opinion score.....	132
Compressing Content from an Origin Web Server.....	135
Overview: Enabling content compression from an origin web server.....	135
Accelerating Responses with Metadata Cache Responses.....	137
Overview: Using Metadata cache responses to accelerate responses.....	137
Advanced Metadata Cache Options for General Options	137
Accelerating Traffic with a Local Traffic Policy.....	139
About classifying types of HTTP traffic with a local traffic policy.....	139
Local traffic policy matching Strategies settings.....	139
Local traffic policy matching Requires profile settings.....	140
Local traffic policy matching Controls settings.....	140
Local traffic policy matching condition types	140
Local traffic policy matching Actions operands	143
Accelerating Traffic with Intelligent Client Cache.....	147
About intelligent client cache.....	147
Using Forward Error Correction to Mitigate Packet Loss.....	149
Overview: Using forward error correction (FEC) to mitigate packet loss.....	149
About forward error correction (FEC).....	149
Using the Request Logging Profile.....	151
Overview: Configuring a Request Logging profile.....	151
About the Request Logging profile.....	151
Standard log formats.....	151
Request Logging profile settings.....	153
Request Logging parameters.....	154
Monitoring BIG-IP Acceleration Application Performance.....	157

Overview: Monitoring the performance of a BIG-IP acceleration application.....	157
Advanced performance monitor settings for general options	157
Overview: ROI reports.....	157
About Byte Savings reports.....	158
About Caching Requests Saved reports.....	158
About IBR Savings reports.....	158
About Inlined Links reports.....	159
About ICC Savings reports.....	159
Managing Deduplication.....	161
What is symmetric data deduplication?.....	161
Which codec do I choose?.....	161
About Discovery.....	163
About discovery on BIG-IP AAM systems.....	163
About subnet discovery.....	163
About dynamic discovery of remote endpoints.....	163
Legal Notices.....	165
Legal notices.....	165

Introducing Acceleration

Overview: Introduction to acceleration

BIG-IP® acceleration, deployed symmetrically, asymmetrically, or in combination, can significantly improve transaction response times. It includes specific techniques that modify or optimize the way in which TCP and other protocols, applications, and data flows can function across the network. Acceleration features enable you to refine transaction response times, according to your specific needs.

Acceleration feature	Benefit
Origin web server load balancing	Enables users to access the best-performing source.
Global server load balancing	Enables users to access the best-performing site.
Compression	Reduces the amount of transmitted data. Asymmetric compression condenses web data for transmission to a browser. Symmetric compression condenses any data transmission to a remote acceleration device.
Data deduplication	Replaces previously sent data with dictionary pointers to minimize transmitted data and improve response time. Also ensures that the data is current and delivered only to authorized users. This feature is available only with a licensed BIG-IP® device.
TCP optimization	Improves TCP performance. Asymmetric optimization aggregates requests for any TCP protocol to reduce connection processing. It optimizes TCP processing for TCP/IP stacks that increase client-side connections to speed web page downloads. Symmetric optimization aggregates transactions inside tunnels that connect acceleration devices.
Web browser object caching	Manipulates HTTP responses to increase browser caching and decrease HTTP requests.
Remote web object caching	Reduces client response times by serving web objects directly from a remote device, rather than from a central server.
HTTP protocol and web application optimization	Manipulates web requests and responses to increase HTTP and web application efficiency.

Origin web server load balancing

A virtual IP address can be configured on a BIG-IP® Local Traffic Manager™, which then acts as a proxy for that IP address. The BIG-IP Local Traffic Manager directs a client request made to the VIP to a server among a pool of servers, all of which are configured to respond to requests made to that address. A server pool improves the response time by reducing the load on each server and, consequently, the time required to process a request.

About data centers

All of the resources on your network are associated with a data center. BIG-IP® DNS consolidates the paths and metrics data collected from the servers, virtual servers, and links in the data center. BIG-IP DNS uses that data to conduct load balancing and route client requests to the best-performing resource based on different factors.

BIG-IP DNS might send all requests to one data center when another data center is down. Alternatively, BIG-IP DNS might send a request to the data center that has the fastest response time. A third option might be for BIG-IP DNS to send a request to the data center that is located closest to the client's source address.

Tip: The resources associated with a data center are available only when the data center is also available.

Data compression

Compression of HTTP and HTTPS traffic removes repetitive data and reduces the amount of data transmitted. Compression provided by the Local Traffic Manager™ offloads the compression overhead from origin web servers and allows the Local Traffic Manager to perform other optimizations that improve performance for an HTTP or HTTPS stream.

Data deduplication

Data deduplication requires the symmetric acceleration provided by BIG-IP® Application Acceleration Manager™ (AAM™). A client-side device sends a request to a server-side device. The server-side device responds to the client object request by sending new data and a dictionary entry or pointer that refers to the data to the client-side device. The client-side device stores the data and the pointer before sending it on to the requesting client. When a user requests the data a second or subsequent time from the client-side device, the server-side device checks for changes to the data, and then sends one or more pointers and any new data that has not been previously sent.

Optimization of TCP connections

The BIG-IP® application acceleration provides MultiConnect functionality that decreases the number of server-side TCP connections required while increasing the number of simultaneous client-side TCP connections available to a browser for downloading a web page.

Decreasing the number of server-side TCP connections can improve application performance and reduce the number of servers required to host an application. Creating and closing a TCP connection requires significant overhead, so as the number of open server connections increases, maintaining those connections while simultaneously opening new connections can severely degrade server performance and user response time.

Despite the ability for multiple transactions to occur within a single TCP connection, a connection is typically between one client and one server. A connection normally closes either when a server reaches a defined transaction limit or when a client has transferred all of the files that are needed from that server. The BIG-IP system, however, operates as a proxy and can pool TCP server-side connections by combining many separate transactions, potentially from multiple users, through fewer TCP connections. The BIG-IP system opens

new server-side connections only when necessary, thus reusing existing connections for requests from other users whenever possible.

The **Enable MultiConnect To** check box on the **Assembly** screen of BIG-IP applies MultiConnect functionality to image or script objects that match the node. The **Enable MultiConnect Within** check box, however, applies MultiConnect functionality to image or script objects that are linked within HTML or CSS files for the node.

Caching objects

Caching provides storage of data within close proximity of the user and permits reuse of that data during subsequent requests.

Web browser objects

In one form of caching, a BIG-IP® instructs a client browser to cache an object, marked as static, for a specified period. During this period, the browser reads the object from cache when building a web page until the content expires, whereupon the client reloads the content. This form of caching enables the browser to use its own cache instead of expending time and bandwidth by accessing data from a central site.

Remote web objects

In a second form of caching, a BIG-IP in a data center manages requests for web application content from origin web servers. Operating asymmetrically, the BIG-IP caches objects from origin web servers and delivers them directly to clients. The BIG-IP module handles both static content and dynamic content, by processing HTTP responses, including objects referenced in the response, and then sending the included objects as a single object to the browser. This form of caching reduces server TCP and application processing, improves web page loading time, and reduces the need to regularly expand the number of web servers required to service an application.

Non-web objects

In a third form of caching, a BIG-IP at a remote site, operating symmetrically, caches and serves content to users. The BIG-IP serves content locally whenever possible, thus reducing the response time and use of the network.

Optimization of HTTP protocol and web applications

HTTP protocol optimization achieves a high user performance level by optimally modifying each HTTP session. Some web applications, for example, cannot return an HTTP 304 status code (Not Modified) in response to a client request, consequently returning an object. Because the BIG-IP® proxies connections and caches content, when a requested object is unchanged, the BIG-IP returns an HTTP 304 response instead of returning the unchanged object, thus enabling the browser to load the content from its own cache even when a web application hard codes a request to resend the object.

The BIG-IP improves the performance of Web applications by modifying server responses, which includes marking an object as cacheable with a realistic expiration date. This optimization is especially beneficial when using off-the-shelf or custom applications that impede or prevent changes to code.

Overview: BIG-IP Acceleration

The BIG-IP® provides an acceleration delivery solution designed to improve the speed at which users access your web applications (such as Microsoft® SharePoint, Microsoft® Outlook Web Access, BEA AquaLogic®, SAP® Portal, Oracle® Siebel™ CRM, Oracle® Portal, and others) and wide area network (WAN).

The BIG-IP accelerates access to your web applications by using acceleration policy features that modify web browser behavior, as well as compress and cache dynamic and static content, decreasing bandwidth usage and ensuring that your users receive the most expedient and efficient access to your web applications and WAN.

Important: *Transparent Data Reduction™ (TDR™) functionality (TDR2 and TDR3) is not supported on a BIG-IP Virtual Edition (VE) or VIPRION® system.*

Application management

To accelerate and manage access to your applications, the BIG-IP® system uses acceleration policies to manipulate HTTP responses from origin web servers. After the BIG-IP system manipulates the HTTP responses, it processes the responses. Therefore, the BIG-IP system processes manipulated responses, rather than the original responses that are sent by the origin web servers.

Application monitoring

You can easily monitor your HTTP traffic and system processes by using various monitoring tools. You can use BIG-IP® application acceleration performance reports, the BIG-IP Dashboard, and other related statistics or logging, as necessary to acquire the information that you want.

Deployment of Distributed BIG-IP Application Acceleration

You can deploy BIG-IP application acceleration functionality to optimize HTTP traffic in a worldwide, geographically distributed configuration.

A geographically distributed deployment consists of two or more BIG-IP devices installed on each end of a WAN: one in the same location as the origin web servers that are running the applications to which the BIG-IP is accelerating client access, and the other near the clients initiating the requests.

Deploying multiple BIG-IP devices achieves additional flexibility in controlling where processing takes place. To prevent sending assembled documents over the WAN, all assembly, except PDF and image optimization, occurs on the device that resides closest to the initiated request.

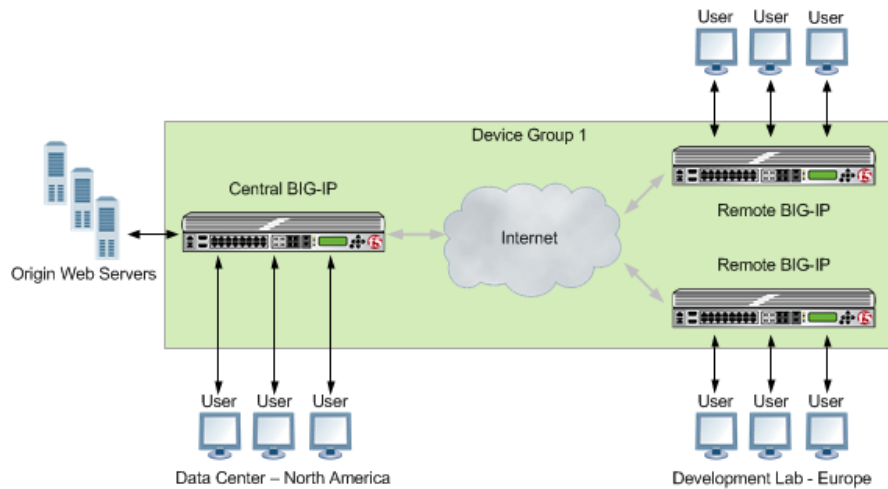


Figure 1: A distributed application acceleration deployment

Management of requests to origin web servers

Most sites are built on a collection of web servers, application servers, and database servers, together known as origin web servers. The BIG-IP® system is installed on your network between the users of your applications and the origin web servers on which the applications run, and accelerates your application's response to HTTP requests.

Origin web servers can serve all possible permutations of content, while the BIG-IP system only stores and serves page content that clients have previously requested from your site. By transparently servicing the bulk of common requests, the BIG-IP system significantly reduces the load on your origin web servers, which improves performance for your site.

Once installed, the BIG-IP system receives all requests destined for the origin web server. When a client makes an initial request for a specific object, the BIG-IP system relays the request to the origin web server, and caches the response that it receives in accordance with the policy, before forwarding the response to the client. The next time a client requests the same object, the BIG-IP system serves the response from cache, based on lifetime settings within the policy, instead of sending the request to the origin web servers.

For each HTTP request that the BIG-IP system receives, the system performs one of the following actions.

Action	Description
Serves the request from its cache	Upon receiving a request from a browser or web client, the BIG-IP system initially checks to see if it can service the request from compiled responses in the system's cache.
Sends the request to the origin web servers	If the BIG-IP system is unable to service the request from the system's cache, it sends a request to the origin web server. Once it receives a response from the origin web server, the BIG-IP system caches that response according to the associated acceleration policy rules, and then forwards the request to the client.
Relays the request to the origin web servers	The BIG-IP system relays requests directly to the origin web server, for some predefined types of content, such as requests for streaming video.
Creates a tunnel to send the request to the origin web servers	For any encrypted traffic (HTTPS) content that you do not want the BIG-IP system to process, you can use tunneling. Note that the BIG-IP system can cache and respond to SSL traffic without using tunnels.

During the process of *application matching*, the BIG-IP system uses the hostname in the HTTP request to match the request to an application profile that you created. Once matched to an application profile, the BIG-IP system applies the associated acceleration policy's *matching rules* in order to group the request and response to a specific leaf node on the Policy Tree. The BIG-IP system then applies the acceleration policy's acceleration rules to each group. These *acceleration rules* dictate how the BIG-IP system manages the request.

Management of responses to clients

The first time that a BIG-IP® system receives new content from the origin web server in response to an HTTP request, it completes the following actions, before returning the requested object (response) to the client.

Action	Description
Compiles an internal representation of the object	The BIG-IP system uses compiled responses received from the origin web server, to assemble an object in response to an HTTP request.
Assigns a Unique Content Identifier (UCI) to the compiled response, based on elements present in the request	The origin web server generates specific responses based on certain elements in the request, such as the URI and query parameters. The BIG-IP system includes these elements in a UCI that it creates, so that it can easily match future requests to the correct content in its cache. The BIG-IP system matches content to the UCI for both the request and the compiled response that it created to service the request.

Flow of requests and responses

The BIG-IP® system processes application requests and responses in a general sequential pattern.

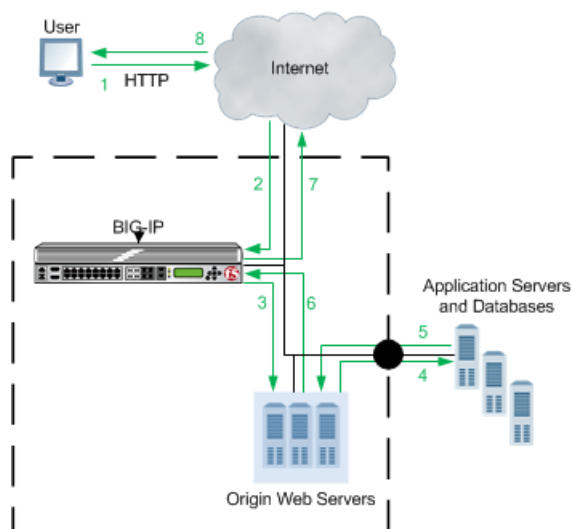


Figure 2: Request and response flow

Each step is processed in the following sequence.

1. Clients, using web browsers, request pages from your site. From the client's perspective, they are connecting directly to your site; they have no knowledge of the BIG-IP system.
2. The BIG-IP system examines the client's request to determine if it meets all the HTTP requirements needed to service the request. If the request does not meet the HTTP requirements, the BIG-IP system issues an error to the client.
3. The BIG-IP system examines the request elements and creates a UCI, and then reviews the system's cache to see if it has a compiled response stored under that same UCI.

If the content is being requested for the first time (there is no matching compiled response in the system's cache), the BIG-IP system uses the host map to relay the request to the appropriate origin web server to get the required content.

If content with the same UCI is already stored as a compiled response in the system's cache, the BIG-IP system checks to see if the content has expired. If the content has expired, the BIG-IP system checks to see if the information in the system's cache still matches the origin web server. If it does, the BIG-IP system moves directly to step 7. Otherwise, it performs the following step.

4. The origin web server either responds or queries the application servers or databases content.
5. The application servers or databases provide the input back to the origin web server.
6. The origin web server replies to the BIG-IP system with the requested material, and the BIG-IP system compiles the response. If the response meets the appropriate requirements, the BIG-IP system stores the compiled response in the system's cache under the appropriate UCI.
7. The BIG-IP system uses the compiled response, and any associated assembly rule parameters, to recreate the page. The assembly rule parameters dictate how to update the page with generated content.
8. The BIG-IP system directs the response to the client.

About symmetric optimization using iSession on BIG-IP systems

The BIG-IP® systems work in pairs on opposite sides of the WAN to optimize the traffic that flows between them through an iSession™ connection. A simple point-to-point configuration might include BIG-IP systems in data centers on opposite sides of the WAN. Other configuration possibilities include point-to-multipoint (also called hub and spoke) and mesh deployments.

The following illustration shows an example of the flow of traffic across the WAN through a pair of BIG-IP devices. In this example, traffic can be initiated on both sides of the WAN.

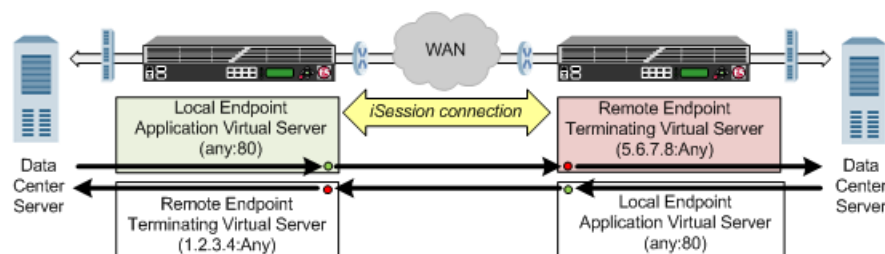


Figure 3: Example of traffic flow through a BIG-IP pair with iSession connection

Each BIG-IP device is an *endpoint*. From the standpoint of each BIG-IP device, it is the *local endpoint*. Any BIG-IP device with which the local endpoint interacts is a *remote endpoint*. After you identify the endpoints, communication between the BIG-IP pair takes place in an iSession connection between the two devices. When you configure the local BIG-IP device, you also identify any *advertised routes*, which are subnets that can be reached through the local endpoint. When viewed on a remote system, these subnets appear as *remote advertised routes*.

To optimize traffic, you create iApps™ templates to select the applications you want to optimize, and the BIG-IP system sets up the necessary virtual servers and associated profiles. The system creates a virtual

server on the initiating side of the WAN, with which it associates a profile that listens for TCP traffic of a particular type (HTTP, CIFS, FTP). The local BIG-IP system also creates a virtual server, called an *iSession listener*, to receive traffic from the other side of the WAN, and it associates a profile that terminates the iSession connection and forwards the traffic to its destination. For some applications, the system creates an additional virtual server to further process the application traffic.

The default iSession profile, which the system applies to application optimization, includes symmetric adaptive compression and symmetric data deduplication.

Accelerating Traffic with Acceleration Profiles

About HTTP compression profiles

HTTP compression reduces the amount of data to be transmitted, thereby significantly reducing bandwidth usage. All of the tasks needed to configure HTTP compression on the BIG-IP® system, as well as the compression software itself, are centralized on the BIG-IP system. The tasks needed to configure HTTP compression for objects in an Application Acceleration Manager module policy node are available in the Application Acceleration Manager, but an HTTP compression profile must be enabled for them to function.

When configuring the BIG-IP system to compress data, you can:

- Configure the system to include or exclude certain types of data.
- Specify the levels of compression quality and speed that you want.

You can enable the HTTP compression option by setting the **URI Compression** or the **Content Compression** setting of the **HTTP Compression** profile to **URI List** or **Content List**, respectively. This causes the BIG-IP system to compress HTTP content for any responses in which the values that you specify in the **URI List** or **Content List** settings of an HTTP profile match the values of the `Request-URI` or `Content-Type` response headers.

Exclusion is useful because some URI or file types might already be compressed. Using CPU resources to compress already-compressed data is not recommended because the cost of compressing the data usually outweighs the benefits. Examples of regular expressions that you might want to specify for exclusion are `.*\.pdf`, `.*\.gif`, or `.*\.html`.

Note: The string that you specify in the **URI List** or the **Content List** setting can be either a pattern string or a regular expression. List types are case-sensitive for pattern strings. For example, the system treats the pattern string `www.f5.com` differently from the pattern string `www.F5.com`. You can override this case-sensitivity by using the Linux `regex` command.

HTTP Compression profile options

You can use an HTTP Compression profile alone, or with the BIG-IP® Application Acceleration Manager, to reduce the amount of data to be transmitted, thereby significantly reducing bandwidth usage. The tasks needed to configure HTTP compression for objects in an Application Acceleration Manager policy node are available in the Application Acceleration Manager, but an HTTP Compression profile must be enabled for them to function.

About Web Acceleration profiles

When used by the BIG-IP system without an associated Application Acceleration Manager application, the Web Acceleration profile uses basic default acceleration.

When used with the Application Acceleration Manager, the Web Acceleration profile includes an ordered list of associated Application Acceleration Manager applications, each of which defines the host names, IP

addresses, and policy that is applied to a request that matches the specified host name or IP address. You can enable one or more Application Acceleration Manager applications in a Web Acceleration profile.

A Web Acceleration profile with multiple Application Acceleration Manager applications that target different host names can be handled by the same virtual server, or by multiple virtual servers, while simultaneously allowing each application to apply a different policy to matching traffic.

The Application Acceleration Manager is enabled by configuring an Application Acceleration Manager application and enabling it in the Web Acceleration profile.

Web Acceleration profile settings

This table describes the Web Acceleration profile configuration settings and default values.

Setting	Value	Description
Name	No default	Specifies the name of the profile.
Parent Profile	Selected predefined or user-defined profile	Specifies the selected predefined or user-defined profile.
Partition / Path	Common	Specifies the partition and path to the folder for the profile objects.
Cache Size	100	Without a provisioned BIG-IP® Application Acceleration Manager, this setting specifies the maximum size in megabytes (MB) reserved for the cache. When the cache reaches the maximum size, the system starts removing the oldest entries. With a provisioned Application Acceleration Manager, this setting defines the minimum reserved cache size. The maximum size of the minimum reserved cache is 64 GB (with provisioned cache availability). An allocation of 15 GB is practical for most implementations. The total available cache includes the minimum reserved cache and a dynamic cache, used as necessary when the minimum reserved cache is exceeded, for a total cache availability of 256 GB.
Maximum Entries	10000	Specifies the maximum number of entries that can be in the cache.
Maximum Age	3600	Specifies how long in seconds that the system considers the cached content to be valid.
Minimum Object Size	500	Specifies the smallest object in bytes that the system considers eligible for caching.
Maximum Object Size	50000	Specifies the largest object in bytes that the system considers eligible for caching.
URI Caching	Not Configured	Specifies whether the system retains or excludes certain Uniform Resource Identifiers (URIs) in the cache. The process forces the system either to cache URIs that typically are ineligible for caching, or to not cache URIs that typically are eligible for caching.
URI List	No default value	Specifies the URIs that the system either includes in or excludes from caching. <ul style="list-style-type: none"> Pin List. Lists the URIs for responses that you want the system to store indefinitely in the cache.

Setting	Value	Description
		<ul style="list-style-type: none"> Include List. Lists the URIs that are typically ineligible for caching, but the system caches them. Determines if a request should be evaluated normally according to caching rules. Exclude List. Lists the URIs that are typically eligible for caching, but the system does not cache them. Include Override List. Lists URIs to cache, though typically, they would not be cached due to defined constraints, for example, the Maximum Object Size setting. The default value is none. URIs in the Include Override List are cacheable even if they are not specified in the Include List. <p><i>Note:</i> You can use regular expressions to specify URIs in accordance with BIG-IP supported meta characters.</p>
Ignore Headers	All	<p>Specifies how the system processes client-side Cache-Control headers when caching is enabled.</p> <ul style="list-style-type: none"> None. Specifies that the system honors all Cache-Control headers. Cache-Control:max-age. Specifies that the system disregards a Cache-Control:max-age request header that has a value of max-age=0. All. Specifies that the system disregards all Cache-Control headers.
Insert Age Header	Enabled	Specifies, when enabled, that the system inserts Date and Age headers in the cached entry. The Date header contains the current date and time on the BIG-IP® system. The Age header contains the length of time that the content has been in the cache.
Aging Rate	9	Specifies how quickly the system ages a cache entry. The aging rate ranges from 0 (slowest aging) to 10 (fastest aging).
AM Applications	No default	Lists enabled Application Acceleration Manager applications in the Enabled field and available applications in the Available field.

Meta characters

This table describes the meta characters that are supported by the BIG-IP for pattern matching.

Meta character	Description	Example
.	Matches any single character.	
^	Matches the beginning of the line in a regular expression. The BIG-IP assumes that the beginning and end of line meta characters exist for every regular expression it sees.	
\$	Matches the end of the line. The BIG-IP assumes that the beginning and end of line meta characters exist for every regular expression it sees.	<p>The expression G.*P.* matches:</p> <ul style="list-style-type: none"> GrandPlan GreenPeace GParse

Meta character	Description	Example
		<ul style="list-style-type: none"> GP <p>A pattern starting with the * character is the same as using .* For example, the BIG-IP interprets the following two expressions as identical.</p> <ul style="list-style-type: none"> *Plan .*Plan
*	Matches zero or more of the patterns that precede it.	
+	Matches one or more of the patterns that precede it.	<p>The expression G.+P.* matches:</p> <ul style="list-style-type: none"> GrandPlan GreenPeace <p>Do not begin a pattern with the + character. For example, do not use +Plan. Instead, use .+Plan.</p>
?	Matches none, or one of the patterns that precede it.	<p>The expression G.?P.* matches:</p> <ul style="list-style-type: none"> GParse GP <p>Do not begin a pattern with the ? character. For example, do not use ?Plan. Instead, use .?Plan.</p>
[...]	Matches a set of characters. You can list the characters in the set using a string made of the characters to match.	<p>The expression C[AHR] matches:</p> <ul style="list-style-type: none"> CAT CHARISMA CRY <p>You can also provide a range of characters by using a dash. For example, the expression AA[0-9]+ matches:</p> <ul style="list-style-type: none"> AA269812209 AA2 <p>It does not, however, match AAB2.</p> <p>To match any alphanumeric character, both upper-case and lower-case, use the expression [a-zA-Z0-9].</p>
[^...]	Matches any character not in the set. Just as with the character, [...], you can specify the individual characters, or a range of characters by using a dash (-).	<p>The expression C[^AHR].* matches:</p> <ul style="list-style-type: none"> CLEAR CORY CURRENT <p>The expression C[^AHR].*, however, does not match:</p> <ul style="list-style-type: none"> CAT CHARISMA

Meta character	Description	Example
(...)	Matches the regular expression contained inside the parenthesis, as a group.	<ul style="list-style-type: none"> CRY The expression AA(12)+CV matches: <ul style="list-style-type: none"> AA12CV AA121212CV
exp1 exp2	Matches either exp1 or exp2, where exp1 and exp2 are regular expressions.	The expression AA([de]12 [zy]13)CV matches: <ul style="list-style-type: none"> AA12CV AAe12CV AAz12CV AAy13CV

Web Acceleration Profile statistics description

This topic provides a description of Web Acceleration Profile statistics produced in tmsh.

Viewing Web Acceleration profile statistics

Statistics for the Web Acceleration Profile can be viewed in tmsh by using the following command.

```
tmsh show /ltm profile web-acceleration <profile_name>
```

Each statistic is described in the following tables.

Table 1: Virtual server statistics

Statistic	Description
Virtual Server	The name of the associated virtual server.

Table 2: Cache statistics

Statistic	Description
Cache Size (in Bytes)	The cache size for small objects (<4k) in cache and metastor tracking data.
Total Cached Items	The total number of objects cached in the local cache for each TMM.
Total Evicted Items	The total number of small objects (<4k) and metastor tracking data entities evicted from cache.
Inter-Stripe Size (in Bytes)	The inter-stripe cache size for small objects (<4k) in cache and metastor tracking data.
Inter-Stripe Cached Items	The total number of objects in the inter-stripe caches for each TMM.
Inter-Stripe Evicted Items	The total number of small objects (<4k) and metastor tracking data entities evicted from the inter-stripe cache for each TMM.

Table 3: Cache Hits/Misses statistics

Statistic	Description
Hits	The total number of cache hits.
Misses (Cacheable)	The number of cache misses for objects that can otherwise be cached.
Misses (Total)	The number of cache misses for all objects.
Inter-Stripe Hits	The number of inter-stripe cache hits for each TMM.
Inter-Stripe Misses	The number of inter-stripe cache misses for each TMM.
Remote Hits	For LTM only, the number of cache hits for owner TMMs. For AAM, this statistic does not apply. Note that an increment of two can occur on first request.
Remote Misses	The number of cache misses for owner TMMs.

About iSession profiles

The iSession[™] profile tells the system how to optimize traffic. Symmetric optimization requires an iSession profile at both ends of the iSession connection. The system-supplied parent iSession profile `isession`, is appropriate for all application traffic, and other iSession profiles have been pre-configured for specific applications. The name of each pre-configured iSession profile indicates the application for which it was configured, such as `isession-cifs`.

When you configure the iSession local endpoint on the Quick Start screen, the system automatically associates the system-supplied iSession profile `isession` with the iSession listener `isession-virtual` it creates for inbound traffic.

You must associate an iSession profile with any virtual server you create for a custom optimized application for outbound traffic, and with any iSession listener you create for inbound traffic.

Screen capture showing compression settings

The following screen capture shows the pertinent compression settings.

Note: If adaptive compression is disabled, you must manually select a compression codec for iSession[™] traffic. If you leave the other codecs enabled, the BIG-IP[®] system selects the `bzip2` compression algorithm by default, and that might not be the algorithm you want.

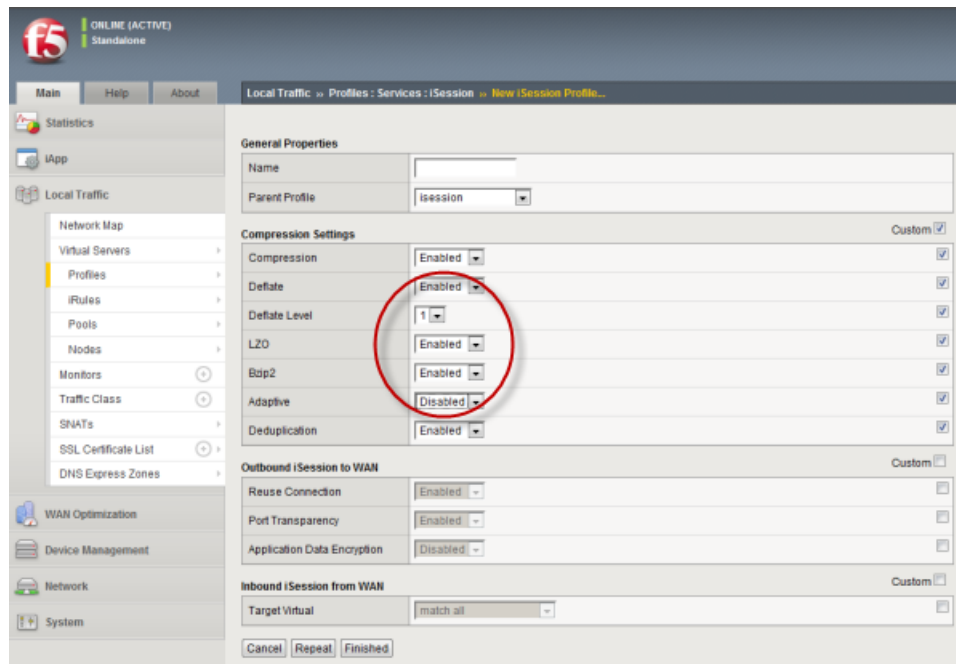


Figure 4: iSession profile screen with compression settings emphasized

About CIFS traffic optimization

Common Internet File System (CIFS) is a remote file access protocol that forms the basis of Microsoft® Windows® file sharing. Various CIFS implementations (for example, Samba) are also available on other operating systems such as Linux™. CIFS is the protocol most often used for transferring files over the network. Using symmetric optimization, the BIG-IP® system can optimize CIFS traffic, resulting in faster performance for transferring CIFS files, opening Microsoft applications, and saving files. CIFS optimization is particularly useful when two offices that are located far apart frequently need to share and exchange files.

Important: By default, Microsoft Windows clients do not require Server Message Block (SMB) signing, except when communicating with their domain controller. If SMB signing settings have been changed, make sure that SMB signing is optional on all servers and clients.

About MAPI optimization

Messaging Application Program Interface (MAPI) is the email protocol that Microsoft® Exchange Server and Outlook® clients use to exchange messages. Optimization of MAPI traffic across the WAN requires a virtual server for each Exchange-based server so that the BIG-IP® system can use the IP addresses of the Exchange-based servers to locate MAPI traffic.

About TCP profiles

TCP profiles are configuration tools that help you to manage TCP network traffic. Many of the configuration settings of TCP profiles are standard SYSCTL types of settings, while others are unique to the BIG-IP® system.

TCP profiles are important because they are required for implementing certain types of other profiles. For example, by implementing TCP, HTTP, Rewrite, HTML, and OneConnect™ profiles, along with a persistence profile, you can take advantage of various traffic management features, such as:

- Content spooling, to reduce server load
- OneConnect, to pool idle server-side connections
- Layer 7 session persistence, such as hash or cookie persistence
- iRules® for managing HTTP traffic
- HTTP data compression
- HTTP pipelining
- URI translation
- HTML content modification
- Rewriting of HTTP redirections

The BIG-IP® system includes several pre-configured TCP profiles that you can use as is. In addition to the default `tcp` profile, the system includes TCP profiles that are pre-configured to optimize LAN and WAN traffic, as well as traffic for mobile users. You can use the pre-configured profiles as is, or you can create a custom profile based on a pre-configured profile and then adjust the values of the settings in the profiles to best suit your particular network environment.

About tcp-lan-optimized profile settings

The `tcp-lan-optimized` profile is a pre-configured profile type that can be associated with a virtual server. In cases where the BIG-IP virtual server is load balancing LAN-based or interactive traffic, you can enhance the performance of your local-area TCP traffic by using the `tcp-lan-optimized` profile.

If the traffic profile is strictly LAN-based, or highly interactive, and a standard virtual server with a TCP profile is required, you can configure your virtual server to use the `tcp-lan-optimized` profile to enhance LAN-based or interactive traffic. For example, applications producing an interactive TCP data flow, such as SSH and TELNET, normally generate a TCP packet for each keystroke. A TCP profile setting such as **Slow Start** can introduce latency when this type of traffic is being processed. By configuring your virtual server to use the **tcp-lan-optimized** profile, you can ensure that the BIG-IP system delivers LAN-based or interactive traffic without delay.

A `tcp-lan-optimized` profile is similar to a TCP profile, except that the default values of certain settings vary, in order to optimize the system for LAN-based traffic.

You can use the `tcp-lan-optimized` profile as is, or you can create another custom profile, specifying the `tcp-lan-optimized` profile as the parent profile.

About tcp-mobile-optimized profile settings

The `tcp-mobile-optimized` profile is a pre-configured profile type, for which the default values are set to give better performance to service providers' 3G and 4G customers. Specific options in the pre-configured profile are set to optimize traffic for most mobile users, and you can tune these settings to fit your network.

For files that are smaller than 1 MB, this profile is generally better than the `mptcp-mobile-optimized` profile. For a more conservative profile, you can start with the `tcp-mobile-optimized` profile, and adjust from there.

Note: *Although the pre-configured settings produced the best results in the test lab, network conditions are extremely variable. For the best results, start with the default settings and then experiment to find out what works best in your network.*

This list provides guidance for relevant settings

- Set the **Proxy Buffer Low** to the **Proxy Buffer High** value minus 64 KB. If the **Proxy Buffer High** is set to less than 64K, set this value at 32K.
- The size of the **Send Buffer** ranges from 64K to 350K, depending on network characteristics. If you enable the **Rate Pace** setting, the send buffer can handle over 128K, because rate pacing eliminates some of the burstiness that would otherwise exist. On a network with higher packet loss, smaller buffer sizes perform better than larger. The number of loss recoveries indicates whether this setting should be tuned higher or lower. Higher loss recoveries reduce the goodput.
- Setting the **Keep Alive Interval** depends on your fast dormancy goals. The default setting of 1800 seconds allows the phone to enter low power mode while keeping the flow alive on intermediary devices. To prevent the device from entering an idle state, lower this value to under 30 seconds.
- The **Congestion Control** setting includes delay-based and hybrid algorithms, which might better address TCP performance issues better than fully loss-based congestion control algorithms in mobile environments. The Illinois algorithm is more aggressive, and can perform better in some situations, particularly when object sizes are small. When objects are greater than 1 MB, goodput might decrease with Illinois. In a high loss network, Illinois produces lower goodput and higher retransmissions.
- For 4G LTE networks, specify the **Packet Loss Ignore Rate** as 0. For 3G networks, specify 2500. When the **Packet Loss Ignore Rate** is specified as more than 0, the number of retransmitted bytes and receives SACKs might increase dramatically.
- For the **Packet Loss Ignore Burst** setting, specify within the range of 6–12, if the **Packet Loss Ignore Rate** is set to a value greater than 0. A higher **Packet Loss Ignore Burst** value increases the chance of unnecessary retransmissions.
- For the **Initial Congestion Window Size** setting, round trips can be reduced when you increase the initial congestion window from 0 to 10 or 16.
- Enabling the **Rate Pace** setting can result in improved goodput. It reduces loss recovery across all congestion algorithms, except Illinois. The aggressive nature of Illinois results in multiple loss recoveries, even with rate pacing enabled.

A `tcp-mobile-optimized` profile is similar to a TCP profile, except that the default values of certain settings vary, in order to optimize the system for mobile traffic.

You can use the `tcp-mobile-optimized` profile as is, or you can create another custom profile, specifying the `tcp-mobile-optimized` profile as the parent profile.

About mptcp-mobile-optimized profile settings

The `mptcp-mobile-optimized` profile is a pre-configured profile type for use in reverse proxy and enterprise environments for mobile applications that are front-ended by a BIG-IP® system. This profile provides a more aggressive starting point than the `tcp-mobile-optimized` profile. It uses newer congestion control algorithms and a newer TCP stack, and is generally better for files that are larger than 1 MB. Specific options in the pre-configured profile are set to optimize traffic for most mobile users in this environment, and you can tune these settings to accommodate your network.

***Note:** Although the pre-configured settings produced the best results in the test lab, network conditions are extremely variable. For the best results, start with the default settings and then experiment to find out what works best in your network.*

The enabled **Multipath TCP** (MPTCP) option enables multiple client-side flows to connect to a single server-side flow in a forward proxy scenario. MPTCP automatically and quickly adjusts to congestion in the network, moving traffic away from congested paths and toward uncongested paths.

The **Congestion Control** setting includes delay-based and hybrid algorithms, which can address TCP performance issues better than fully loss-based congestion control algorithms in mobile environments. Refer to the online help descriptions for assistance in selecting the setting that corresponds to your network conditions.

The enabled **Rate Pace** option mitigates bursty behavior in mobile networks and other configurations. It can be useful on high latency or high BDP (bandwidth-delay product) links, where packet drop is likely to be a result of buffer overflow rather than congestion.

An `mptcp-mobile-optimized` profile is similar to a TCP profile, except that the default values of certain settings vary, in order to optimize the system for mobile traffic.

You can use the `mptcp-mobile-optimized` profile as is, or you can create another custom profile, specifying the `mptcp-mobile-optimized` profile as the parent profile.

About tcp-wan-optimized profile settings

The `tcp-wan-optimized` profile is a pre-configured profile type. In cases where the BIG-IP system is load balancing traffic over a WAN link, you can enhance the performance of your wide-area TCP traffic by using the `tcp-wan-optimized` profile.

If the traffic profile is strictly WAN-based, and a standard virtual server with a TCP profile is required, you can configure your virtual server to use a `tcp-wan-optimized` profile to enhance WAN-based traffic. For example, in many cases, the client connects to the BIG-IP virtual server over a WAN link, which is generally slower than the connection between the BIG-IP system and the pool member servers. By configuring your virtual server to use the `tcp-wan-optimized` profile, the BIG-IP system can accept the data more quickly, allowing resources on the pool member servers to remain available. Also, use of this profile can increase the amount of data that the BIG-IP system buffers while waiting for a remote client to accept that data. Finally, you can increase network throughput by reducing the number of short TCP segments that the BIG-IP® system sends on the network.

A `tcp-wan-optimized` profile is similar to a TCP profile, except that the default values of certain settings vary, in order to optimize the system for WAN-based traffic.

You can use the `tcp-wan-optimized` profile as is, or you can create another custom profile, specifying the `tcp-wan-optimized` profile as the parent profile.

About HTTP2 profiles

You can configure the BIG-IP® Acceleration HTTP/2 profile to provide gateway functionality for HTTP 2.0 traffic, minimizing the latency of requests by multiplexing streams and compressing headers.

A client initiates an HTTP/2 request to the BIG-IP system, the HTTP/2 virtual server receives the request on port 443, and sends the request to the appropriate server. When the server provides a response, the BIG-IP system compresses and caches it, and sends the response to the client.

Important: The BIG-IP system supports HTTP/2 for client-side connections only. This means that when a client that supports HTTP/2 connects to a virtual server that has an HTTP/2 profile assigned to it, the resulting server-side traffic (such as traffic sent to pool members) is sent over HTTP/1.1.

Source address persistence is not supported by the HTTP/2 profile.

Summary of HTTP/2 profile functionality

By using the HTTP/2 profile, the BIG-IP system provides the following functionality for HTTP/2 requests.

Creating concurrent streams for each connection.

You can specify the maximum number of concurrent HTTP requests that are accepted on a HTTP/2 connection. If this maximum number is exceeded, the system closes the connection.

Limiting the duration of idle connections.

You can specify the maximum duration for an idle HTTP/2 connection. If this maximum duration is exceeded, the system closes the connection.

Enabling a virtual server to process HTTP/2 requests.

You can configure the HTTP/2 profile on the virtual server to receive HTTP, SPDY, and HTTP/2 traffic, or to receive only HTTP/2 traffic, based in the activation mode you select. (Note the HTTP/2 profile to receive only HTTP/2 traffic is primarily intended for troubleshooting.)

Inserting a header into the request.

You can insert a header with a specific name into the request. The default name for the header is X-HTTP/2.

Important: The HTTP/2 protocol is incompatible with NTLM protocols. Do not use the HTTP/2 protocol with NTLM protocols.

About HTTP/2 profiles

The BIG-IP® system's Acceleration functionality includes an HTTP/2 profile type that you can use to manage HTTP/2 traffic, improving the efficiency of network resources while reducing the perceived latency of requests and responses. The Acceleration HTTP/2 profile enables you to achieve these advantages by multiplexing streams and compressing headers with Transport Layer Security (TLS) or Secure Sockets Layer (SSL) security.

The HTTP/2 protocol uses a binary framing layer that defines a frame type and purpose in managing requests and responses. The binary framing layer determines how HTTP messages are encapsulated and transferred between the client and server, a significant benefit of HTTP 2.0 when compared to earlier versions.

All HTTP/2 communication occurs by means of a connection with bidirectional streams. Each stream includes messages, consisting of one or more frames, that can be interleaved and reassembled using the embedded stream identifier within each frame's header. The HTTP/2 profile enables you to specify a maximum frame size and write size, which controls the total size of combined data frames, to improve network utilization.

Multiplexing streams

You can use the HTTP/2 profile to multiplex streams (interleaving and reassembling the streams), by specifying a maximum number of concurrent streams permitted for a single connection. Also, because multiplexing streams on a single TCP connection compete for shared bandwidth, you can use the profile's Priority Handling settings to configure stream prioritization and define the relative order of delivery. For example, a Strict setting processes higher priority streams to completion before processing lower priority

streams; whereas, a Fair setting allows higher priority streams to use more bandwidth than lower priority streams, without completely blocking the lower priority streams.

Additionally, you can specify the way that the HTTP/2 profile controls the flow of streams. The Receive Window setting allows HTTP/2 to stall individual upload streams, as needed. For example, if the BIG-IP system is unable to process a slow stream on a connection, but is able to process other streams on the connection, it can use the Receive Window setting to specify a frame size for the slow stream, thus delaying that upload stream until the size is met and the receiver is able to process it, while concurrently proceeding to process frames for another stream.

Compressing headers

When you configure the HTTP/2 profile's Header Table Size setting, you can compress HTTP headers to conserve bandwidth. Compressing HTTP headers reduces the object size, which reduces required bandwidth. For example, you can specify a larger table value for better compression, but at the expense of using more memory.

HTTP/2 profile settings

This table provides descriptions of the HTTP/2 profile settings.

Setting	Default	Description
Name		Specifies the name of the HTTP/2 profile.
Parent Profile	http2	Specifies the profile that you want to use as the parent profile. Your new profile inherits all settings and values from the parent profile specified.
Concurrent Streams Per Connection	10	Specifies the number of concurrent requests allowed to be outstanding on a single HTTP/2 connection.
Connection Idle Timeout	300	Specifies the number of seconds an HTTP/2 connection is left open idly before it is closed.
Insert Header	Disabled	Specifies whether an HTTP header that indicates the use of HTTP/2 is inserted into the request sent to the origin web server.
Insert Header Name	X-HTTP/2	Specifies the name of the HTTP header controlled by the Insert Header setting.
Activation Modes	Select Modes	Specifies how a connection is established as a HTTP/2 connection.
Selected Modes	ALPN NPN	Used only with an Activation Modes selection of Select Modes , specifies the extension, ALPN for HTTP/2 or NPN for SPDY, used in the HTTP/2 profile. The order of the extensions in the Selected Modes Enabled list ranges from most preferred (first) to least preferred (last). Clients typically use the first supported extension. At least one HTTP/2 mode must be included in the Enabled list. The values ALPN and NPN specify that the TLS Application Layer Protocol Negotiation (ALPN) and Next Protocol Negotiation (NPN) will be used to determine whether HTTP/2 or SPDY should be activated. Clients that use TLS, but only support HTTP will work as if HTTP/2 is not present. The value Always specifies that all connections function as HTTP/2 connections. Selecting Always in the Activation Mode list is primarily intended for troubleshooting.
Priority Handling	Strict	Specifies how the HTTP/2 profile handles priorities of concurrent streams within the same connection. Selecting Strict processes higher priority

Setting	Default	Description
		streams to completion before processing lower priority streams. Selecting Fair enables higher priority streams to use more bandwidth than lower priority streams, without completely blocking the lower priority streams.
Receive Window	32	Specifies the <i>receive window</i> , which is HTTP/2 protocol functionality that controls flow, in KB. The receive window allows the HTTP/2 protocol to stall individual upload streams when needed.
Frame Size	2048	Specifies the size of the data frames, in bytes, that the HTTP/2 protocol sends to the client. Larger frame sizes improve network utilization, but can affect concurrency.
Write Size	16384	Specifies the total size of combined data frames, in bytes, that the HTTP/2 protocol sends in a single write function. This setting controls the size of the TLS records when the HTTP/2 protocol is used over Secure Sockets Layer (SSL). A large write size causes the HTTP/2 protocol to buffer more data and improves network utilization.
Header Table Size	4096	Specifies the size of the header table, in KB. The HTTP/2 protocol compresses HTTP headers to save bandwidth. A larger table size allows better compression, but requires more memory.

About SPDY profiles

You can use the BIG-IP® system SPDY (pronounced "speedy") profile to minimize latency of HTTP requests by multiplexing streams and compressing headers. When you assign a SPDY profile to an HTTP virtual server, the HTTP virtual server informs clients that a SPDY virtual server is available to respond to SPDY requests.

When a client sends an HTTP request, the HTTP virtual server manages the request as a standard HTTP request. It receives the request on port 80, and sends the request to the appropriate server. When the server provides a response, the BIG-IP system inserts an HTTP header into the response (to inform the client that a SPDY virtual server is available to handle SPDY requests), compresses and caches it, and sends the response to the client.

A client that is enabled to use the SPDY protocol sends a SPDY request to the BIG-IP system, the SPDY virtual server receives the request on port 443, converts the SPDY request into an HTTP request, and sends the request to the appropriate server. When the server provides a response, the BIG-IP system converts the HTTP response into a SPDY response, compresses and caches it, and sends the response to the client.

Summary of SPDY profile functionality

By using the SPDY profile, the BIG-IP system provides the following functionality for SPDY requests.

Creating concurrent streams for each connection.

You can specify the maximum number of concurrent HTTP requests that are accepted on a SPDY connection. If this maximum number is exceeded, the system closes the connection.

Limiting the duration of idle connections.

You can specify the maximum duration for an idle SPDY connection. If this maximum duration is exceeded, the system closes the connection.

Enabling a virtual server to process SPDY requests.

You can configure the SPDY profile on the virtual server to receive both HTTP and SPDY traffic, or to receive only SPDY traffic, based in the activation mode you select. (Note that setting this to receive only SPDY traffic is primarily intended for troubleshooting.)

Inserting a header into the response.

You can insert a header with a specific name into the response. The default name for the header is X-SPDY.

Important: The SPDY protocol is incompatible with NTLM protocols. Do not use the SPDY protocol with NTLM protocols. For additional details regarding this limitation, please refer to the SPDY specification: <http://dev.chromium.org/spdy/spdy-authentication>.

About NTLM profiles

NT LAN Manager (NTLM) is an industry-standard technology that uses an encrypted challenge/response protocol to authenticate a user without sending the user's password over the network. Instead, the system requesting authentication performs a calculation to prove that the system has access to the secured NTLM credentials. NTLM credentials are based on data such as the domain name and user name, obtained during the interactive login process.

The NTLM profile within the BIG-IP® system optimizes network performance when the system is processing NT LAN Manager traffic. When both an NTLM profile and a OneConnect™ profile are associated with a virtual server, the local traffic management system can take advantage of server-side connection pooling for NTLM connections.

How does the NTLM profile work?

When the NTLM profile is associated with a virtual server and the server replies with the HTTP 401 Unauthorized HTTP response message, the NTLM profile inserts a cookie, along with additional profile options, into the HTTP response. The information is encrypted with a user-supplied passphrase and associated with the serverside flow. Further client requests are allowed to reuse this flow only if they present the NTLMConnPool cookie containing the matching information. By using a cookie in the NTLM profile, the BIG-IP system does not need to act as an NTLM proxy, and returning clients do not need to be re-authenticated.

The NTLM profile works by parsing the HTTP request containing the NTLM type 3 message and securely storing the following pieces of information (aside from those which are disabled in the profile):

- User name
- Workstation name
- Target server name
- Domain name
- Cookie previously set (cookie name supplied in the profile)
- Source IP address

With the information safely stored, the BIG-IP system can then use the data as a key when determining which clientside requests to associate with a particular serverside flow. You can configure this using the NTLM profile options. For example, if a server's resources can be openly shared by all users in that server's domain, then you can enable the Key By NTLM Domain setting, and all serverside flows from the users of the same domain can be pooled for connection reuse without further authentication. Or, if a server's resources can be openly shared by all users originating from a particular IP address, then you can enable the Key By Client IP Address setting and all serverside flows from the same source IP address can be pooled for connection reuse.

About OneConnect profiles

The OneConnect profile type implements the BIG-IP® system's OneConnect feature. This feature can increase network throughput by efficiently managing connections created between the BIG-IP system and back-end pool members. You can use the OneConnect feature with any TCP-based protocol, such as HTTP or RTSP.

How does OneConnect work?

The *OneConnect* feature works with request headers to keep existing server-side connections open and available for reuse by other clients. When a client makes a new connection to a virtual server configured with a OneConnect profile, the BIG-IP system parses the request, selects a server using the load-balancing method defined in the pool, and creates a connection to that server. When the client's initial request is complete, the BIG-IP system temporarily holds the connection open and makes the idle TCP connection to the pool member available for reuse.

When another connection is subsequently initiated to the virtual server, if an existing server-side flow to the pool member is open and idle, the BIG-IP system applies the OneConnect source mask to the IP address in the request to determine whether the request is eligible to reuse the existing idle connection. If the request is eligible, the BIG-IP system marks the connection as non-idle and sends a client request over that connection. If the request is not eligible for reuse, or an idle server-side flow is not found, the BIG-IP system creates a new server-side TCP connection and sends client requests over the new connection.

Note: The BIG-IP system can pool server-side connections from multiple virtual servers if those virtual servers reference the same OneConnect profile and the same pool. Also, the re-use of idle connections can cause the BIG-IP system to appear as though the system is not load balancing traffic evenly across pool members.

About client source IP addresses

The standard address translation mechanism on the BIG-IP system translates only the destination IP address in a request and not the source IP address (that is, the client node's IP address). However, when the OneConnect feature is enabled, allowing multiple client nodes to re-use a server-side connection, the source IP address in the header of each client node's request is always the IP address of the client node that initially opened the server-side connection. Although this does not affect traffic flow, you might see evidence of this when viewing certain types of system output.

The OneConnect profile settings

When configuring a OneConnect profile, you specify this information:

Source mask

The mask applied to the source IP address to determine the connection's eligibility to reuse a server-side connection.

Maximum size of idle connections

The maximum number of idle server-side connections kept in the connection pool.

Maximum age before deletion from the pool

The maximum number of seconds that a server-side connection is allowed to remain before the connection is deleted from the connection pool.

Maximum reuse of a connection

The maximum number of requests to be sent over a server-side connection. This number should be slightly lower than the maximum number of HTTP `Keep-Alive` requests accepted by servers in order to prevent the server from initiating a connection close action and entering the `TIME_WAIT` state.

Idle timeout override

The maximum time that idle server-side connections are kept open. Lowering this value may result in a lower number of idle server-side connections, but may increase request latency and server-side connection rate.

OneConnect and HTTP profiles

Content switching for HTTP requests

When you assign both a OneConnect profile and an HTTP profile to a virtual server, and an HTTP client sends multiple requests within a single connection, the BIG-IP system can process each HTTP request individually. The BIG-IP system sends the HTTP requests to different destination servers as determined by the load balancing method. Without a OneConnect profile enabled for the HTTP virtual server, the BIG-IP system performs load-balancing only once for each TCP connection.

HTTP version considerations

For HTTP traffic to be eligible to use the OneConnect feature, the web server must support HTTP `Keep-Alive` connections. The version of the HTTP protocol you are using determines to what extent this support is available. The BIG-IP system therefore includes a *OneConnect transformations* feature within the HTTP profile, specifically designed for use with HTTP/1.0 which by default does not enable `Keep-Alive` connections. With the OneConnect transformations feature, the BIG-IP system can transform HTTP/1.0 connections into HTTP/1.1 requests on the server side, thus allowing those connections to remain open for reuse.

The two different versions of the HTTP protocol treat `Keep-Alive` connections in these ways:

HTTP/1.1 requests

HTTP `Keep-Alive` connections are enabled by default in HTTP/1.1. With HTTP/1.1 requests, the server does not close the connection when the content transfer is complete, unless the client sends a `Connection: close` header in the request. Instead, the connection remains active in anticipation of the client reusing the same connection to send additional requests. For HTTP/1.1 requests, you do not need to use the OneConnect transformations feature.

HTTP/1.0 requests

HTTP `Keep-Alive` connections are not enabled by default in HTTP/1.0. With HTTP/1.0 requests, the client typically sends a `Connection: close` header to close the TCP connection after sending the request. Both the server and client-side connections that contain the `Connection: close` header are closed once the response is sent. When you assign a OneConnect profile to a virtual server, the BIG-IP system transforms `Connection: close` headers in HTTP/1.0 client-side requests to `X-Connection: close` headers on the server side, thereby allowing a client to reuse an existing connection to send additional requests.

OneConnect and SNATs

When a client makes a new connection to a virtual server that is configured with a OneConnect profile and a source network address translation (SNAT) object, the BIG-IP system parses the HTTP request, selects a server using the load-balancing method defined in the pool, translates the source IP address in the request

to the SNAT IP address, and creates a connection to the server. When the client's initial HTTP request is complete, the BIG-IP system temporarily holds the connection open and makes the idle TCP connection to the pool member available for reuse. When a new connection is initiated to the virtual server, the BIG-IP system performs SNAT address translation on the source IP address and then applies the OneConnect source mask to the translated SNAT IP address to determine whether it is eligible to reuse an existing idle connection.

OneConnect and NTLM profiles

NT Lan Manager (NTLM) HTTP 401 responses prevent the BIG-IP® system from detaching the server-side connection. As a result, a late FIN from a previous client connection might be forwarded to a new client that re-used the connection, causing the client-side connection to close before the NTLM handshake completes. If you prefer NTLM authentication support when using the OneConnect feature, you should configure an NTLM profile in addition to the OneConnect profile.

Managing Traffic with Bandwidth Controllers

Overview: Bandwidth control management

Fine-grained bandwidth control is essential to service providers, large enterprises, and remote access services (RAS) solutions. Bandwidth controllers on the BIG-IP® system can scale easily, work well in a distributed environment, and are easy to configure for various networks. Depending on the type of policy you configure, you can use bandwidth controllers to apply specified rate enforcement to traffic flows or mark traffic that exceeds limits.

Bandwidth control policies can be static or dynamic. Through the user interface (browser or `tmssh` command-line utility), when you apply a bandwidth control policy to a virtual server, packet filter, or route domain, you can apply only one policy at a time, and that is a static policy. Using iRules®, you can combine static and dynamic bandwidth control policies up to eight policies on a connection, but only one of the eight policies can be a dynamic policy. A packet is transmitted only when all the attached policies allow it. The system as a whole supports a maximum of 1024 policies.

Important: *Applying a bandwidth controller policy to a route domain affects all traffic transmitted by the BIG-IP system to VLANs in the route domain, including health monitors and DNS queries.*

Note: *Only static bandwidth control policies support SNMP queries.*

Bandwidth controllers vs. rate shaping

Bandwidth controller is the updated version of rate shaping on the BIG-IP® system. These features are mutually exclusive. You can configure and use either rate shaping or bandwidth controllers, but not both. Bandwidth controllers include distributed control, subscriber fairness, and support for a maximum rate of 320 Gbps. Rate shaping is hierarchical and supports minimum bandwidth (committed information rate), priority, and flow fairness.

About static bandwidth control policies

A *static* bandwidth control policy controls the aggregate rate for a group of applications or a network path. It enforces the total amount of bandwidth that can be used, specified as the maximum rate of the resource you are managing. The rate can be the total bandwidth of the BIG-IP® device, or it might be a group of traffic flows.

You can assign a static bandwidth control policy to traffic using a virtual server or, alternatively, you can assign a static bandwidth control policy to a packet filter or a route domain.

About dynamic bandwidth control policies

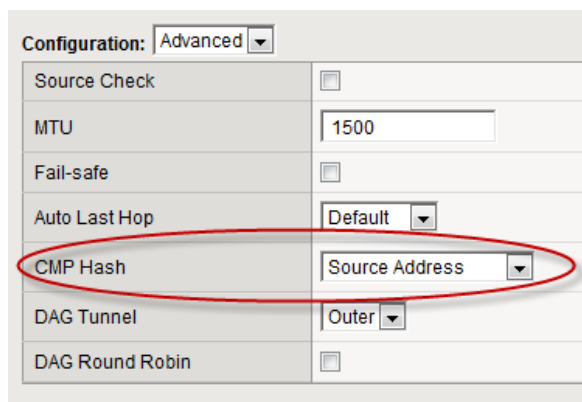
You can create dynamic bandwidth control policies to restrict bandwidth usage per subscriber or group of subscribers, per application, per network egress link, or any combination of these. A *dynamic* bandwidth control policy provides fairness on traffic flows, according to configurable parameters, within an upper bandwidth limit. The BIG-IP® system activates the dynamic bandwidth control policy for each user only when the user participates. When you create a dynamic bandwidth control policy, it acts as a policy in waiting, until the system detects egress traffic that matches the traffic you want to control and creates an instance of the policy. At that moment, the system applies the bandwidth control policy limits, as specified. No bandwidth control occurs until the system detects traffic and creates an instance of the policy. With this feature, an Internet service provider (ISP) can create and revise a single policy that can apply to millions of users.

The BIG-IP system can enforce multiple levels of bandwidth limits through the dynamic policy. For example, a user could be limited by the maximum rate, a per user rate, and a per category rate (such as for an application), all from the same dynamic policy. When the total of the maximum user rate for all the instances exceeds the maximum rate specified in the dynamic policy, the BIG-IP system maintains fairness among all users and spreads the limitation equally among users belonging to a dynamic policy.

You can also configure a dynamic bandwidth control policy to mark packets that exceed the maximum per-user rate for a specified session. The WAN router should handle the marked packets. The BIG-IP system passes packets that conform to the maximum per-user rate without marking them. You configure marking by using the **IP Type of Service** or **Link Quality of Service** setting. For example, a common use of QoS marking is for Voice over IP (VoIP) traffic. VoIP is usually assigned to the Expedited Forwarding (EF) class by using the DSCP value of 46, thus prioritized according to importance and sensitivity to loss/latency. You can mark packets per policy or per category (within a policy). Category marking supersedes policy marking.

The BIG-IP system uses source and destination hashes to control the way incoming traffic is distributed among the instances of the Traffic Management Microkernel (TMM) service. Subscriber-based bandwidth control depends on having a unique one-to-one relationship between bandwidth control policy and subscriber. Subscribers are commonly identified using a unique IP address, and, therefore, load distribution among the instances of TMM service must use the source IP address as the key.

This screen snippet highlights the proper setting.



The screenshot shows a configuration window with a 'Configuration:' dropdown set to 'Advanced'. Below this is a table of settings:

Source Check	<input type="checkbox"/>
MTU	1500
Fail-safe	<input type="checkbox"/>
Auto Last Hop	Default
CMP Hash	Source Address
DAG Tunnel	Outer
DAG Round Robin	<input type="checkbox"/>

The 'CMP Hash' row is circled in red, indicating the correct setting for dynamic bandwidth control.

Figure 5: CMP Hash setting for dynamic bandwidth control

Alternatives for identifying users and applying dynamic bandwidth control policies to traffic are using iRules®, Policy Enforcement Manager™, or Access Policy Manager®.

Example of a dynamic bandwidth control policy

This screen is an example of a dynamic bandwidth control policy that might be created by an Internet service provider (ISP) to manage individual mobile subscribers.

Acceleration » Bandwidth Controllers » **dynamic-policy-1**

⚙️ Properties

General Properties

Name	dynamic-policy-1		
Partition / Path	Common		
Description	Dynamic Policy with total B/W of 200 Mbps		
Maximum Rate	200	Mbps	▼

Dynamic Properties

Dynamic	Enabled		
Maximum Rate Per User	20	Mbps	▼
Measure	Disabled ▼		
Log Period	2048	milliseconds	
Log Publisher			
IP Marking (TOS/DSCP)	Pass Through ▼		
L2 Marking (802.1p)	Pass Through ▼		

Update Clone Delete

Categories

<input checked="" type="checkbox"/>	Name	Description	Maximum Rate (Kbps)	Maximum Rate (%)	IP Marking (TOS/DSCP)	L2 Marking (802.1p)
<input type="checkbox"/>	P2P		0	20	Pass Through	Pass Through
<input type="checkbox"/>	Video		4000	0	Pass Through	Pass Through
<input type="checkbox"/>	VoIP		1000	0	Pass Through	Pass Through
<input type="checkbox"/>	http		0	50	Pass Through	Pass Through

Add Remove

Figure 6: Example of completed dynamic bandwidth control policy screen

In the example, the ISP sets the maximum bandwidth at 200 Mbps. Of that bandwidth, a maximum of 20 Mbps is allocated to each user. Of that allocation, application traffic is apportioned, as follows.

- 20% applies to P2P
- 4 Mbps applies to video
- 1 Mbps applies to Voice over IP (VoIP)
- 50% applies to browser traffic (HTTP)

To activate this policy, the ISP needs to create an iRule to attach the policy to a user session, and then apply the policy to a virtual server.

The bandwidth controller is only an enforcer. For a dynamic bandwidth control policy, you also need iRules®, Policy Enforcement Manager™, or Access Policy Manager® to identify a flow and map it to a category.

Managing Traffic with Rate Shaping

Introduction to rate shaping

The BIG-IP® system includes a feature called rate shaping. *Rate shaping* allows you to enforce a throughput policy on incoming traffic. Throughput policies are useful for prioritizing and restricting bandwidth on selected traffic patterns.

Rate shaping can be useful for an e-commerce site that has preferred clients. For example, the site might want to offer higher throughput for preferred customers, and lower throughput for other site traffic.

The rate shaping feature works by first queuing selected packets under a rate class, and then dequeuing the packets at the indicated rate and in the indicated order specified by the rate class. A *rate class* is a rate-shaping policy that defines throughput limitations and a packet scheduling method to be applied to all traffic handled by the rate class.

You configure rate shaping by creating one or more rate classes and then assigning the rate class to a packet filter or to a virtual server. You can also use the iRules® feature to instruct the BIG-IP system to apply a rate class to a particular connection.

You can apply a rate class specifically to traffic from a server to a client or from a client to a server. If you configure the rate class for traffic that is going to a client, the BIG-IP system does not apply the throughput policy to traffic destined for the server. Conversely, if you configure the rate class for traffic that is going to a server, the BIG-IP system does not apply the throughput policy to traffic destined for the client.

About rate classes

A rate class defines the throughput limitations and packet scheduling method that you want the BIG-IP® system to apply to all traffic that the rate class handles. You assign rate classes to virtual servers and packet filter rules, as well as through iRules®.

If the same traffic is subject to rate classes that you have assigned from more than one location, the BIG-IP system applies the last-assigned rate class only. The BIG-IP system applies rate classes in the following order:

- The first rate class that the BIG-IP system assigns is from the last packet filter rule that matched the traffic and specified a rate class.
- The next rate class that the BIG-IP system assigns is from the virtual server; if the virtual server specifies a rate class, the rate class overrides any rate class that the packet filter selects.
- The last rate class assigned is from the iRule; if the iRule specifies a rate class, this rate class overrides any previously-selected rate class.

Note: *Rate classes cannot reside in partitions. Therefore, a user's ability to create and manage rate classes is defined by user role, rather than partition-access assignment.*

You can create a rate class using the BIG-IP Configuration utility. After you have created a rate class, you must assign it to a virtual server or a packet filter rule, or you must specify the rate class from within an iRule.

Rate class name

The first setting you configure for a rate class is the rate class name. Rate class names are case-sensitive and might contain letters, numbers, and underscores (_) only. Reserved keywords are not allowed.

Each rate class that you define must have a unique name. This setting is required.

To specify a rate class name, locate the `Name` field on the New Rate Class screen and type a unique name for the rate class.

Base rate

The **Base Rate** setting specifies the base throughput rate allowed for traffic that the rate class handles. The sum of the base rates of all child rate classes attached to a parent rate class, plus the base rate of the parent rate class, cannot exceed the ceiling of the parent rate class. For this reason, F5 Networks® recommends that you always set the base rate of a parent rate class to 0 (the default value).

You can specify the base rate in bits per second (bps), kilobits per second (Kbps), megabits per second (Mbps), or gigabits per second (Gbps). The default unit is bits per second. This setting is required.

***Note:** These numbers are powers of 10, not powers of 2.*

Ceiling rate

The **Ceiling Rate** setting specifies the absolute limit at which traffic is allowed to flow when bursting or borrowing. You can specify the ceiling in bits per second (bps), kilobits per second (Kbps), megabits per second (Mbps), or gigabits per second (Gbps). The default unit is bits per second.

If the rate class is a parent rate class, the value of the ceiling defines the maximum rate allowed for the sum of the base rates of all child rate classes attached to the parent rate class, plus the base rate of the parent rate class.

***Note:** A child rate class can borrow from the ceiling of its parent rate class.*

Burst size

You use the **Burst Size** setting when you want to allow the rate of traffic flow that a rate class controls to exceed the base rate. Exceeding the base rate is known as *bursting*. When you configure a rate class to allow bursting (by specifying a value other than 0), the BIG-IP® system saves any unused bandwidth and uses that bandwidth later to enable the rate of traffic flow to temporarily exceed the base rate. Specifying a burst size is useful for smoothing out traffic patterns that tend to fluctuate or exceed the base rate, such as HTTP traffic.

The value of the **Burst Size** setting defines the maximum number of bytes that you want to allow for bursting. Thus, if you set the burst size to 5,000 bytes, and the rate of traffic flow exceeds the base rate by 1,000 bytes per second, then the BIG-IP system allows the traffic to burst for a maximum of five seconds.

When you specify a burst size, the BIG-IP system creates a burst reservoir of that size. A burst reservoir stores bandwidth that the BIG-IP system uses for bursting later. The burst reservoir becomes depleted as the rate of traffic flow exceeds the base rate, and is replenished as the rate of traffic falls below the base rate. The Burst Size value that you configure in a rate class thus represents:

- The maximum number of bytes that the rate class transmits when the traffic-flow rate exceeds the base rate
- The maximum number of bytes that the BIG-IP system can replenish into the burst reservoir
- The amount of bandwidth initially available for bursting beyond the base rate

The burst size is measured in bytes. For example, a value of either 10000 or 10K equals 10,000 bytes. The default value is 0.

Depleting the burst reservoir

When the rate of traffic flow exceeds the base rate, the BIG-IP® system automatically depletes the burst reservoir, at a rate determined by the number of bytes per second that the traffic flow exceeds the base rate.

Continuing with our previous example in which traffic flow exceeds the base rate by 1,000 bytes per second, if the traffic-flow rate only exceeds the base rate for two seconds, then 2,000 bytes are depleted from the burst size and the maximum bytes available for bursting decreases to 3,000.

***Note:** In some cases, a rate class can borrow bandwidth from the burst reservoir of its parent class.*

Replenishing a burst reservoir

When the rate of traffic flow falls below the base rate, the BIG-IP® system stores the unused bandwidth (that is, the difference between the base rate and the actual traffic-flow rate) in the burst reservoir. Later, the BIG-IP system uses this bandwidth when traffic flow exceeds the base rate. Thus, the BIG-IP system replenishes the burst reservoir whenever it becomes depleted due to traffic flow exceeding the base rate.

The size of the burst reservoir cannot exceed the specified burst size. For this reason, the BIG-IP system replenishes the reservoir with unused bandwidth only until the reservoir reaches the amount specified by the **Burst Size** setting. Thus, if the burst size is set to 5,000, then the BIG-IP system can store only 5,000 bytes of unused bandwidth for later use when the rate of traffic flow exceeds the base rate.

***Note:** Specifying a burst size does not allow the rate class to exceed its ceiling.*

About specifying a non-zero burst size

This example illustrates the behavior of the BIG-IP® system when you set the **Burst Size** setting to a value other than 0.

This example shows throughput rates in units of bytes-per-second instead of the default bits-per-second. This is only to simplify the example. You can derive bytes-per-second from bits-per-second by dividing the bits-per-second amount by 8.

Suppose you configure the rate class settings with these values:

- Base rate: 1,000 bytes per second
- Ceiling rate: 4,000 bytes per second
- Burst size: 5,000 bytes

Consider the following scenario:

If traffic is currently flowing at 800 bytes per second

No bursting is necessary because the rate of traffic flow is below the base rate defined in the rate class. Because the traffic is flowing at 200 bytes per second less than the base rate, the BIG-IP system can potentially add 200 bytes of unused bandwidth to the burst reservoir. However, because no bursting has occurred yet, the reservoir is already full at the specified 5,000 bytes, thus preventing the BIG-IP system from storing the 200 bytes of unused bandwidth in the reservoir. In this case, the BIG-IP system simply discards the unused bandwidth.

If traffic climbs to 1,000 bytes per second (equal to the base rate)

Still no bursting occurs, and there is no unused bandwidth.

If traffic jumps to 2,500 bytes per second

For each second that the traffic continues to flow at 2,500 bytes per second, the BIG-IP system empties 1,500 bytes from the burst reservoir (the difference between the traffic flow rate and the base rate). This allows just over three seconds of bursting at this rate before the burst reservoir of 5,000 bytes is depleted. Once the reservoir is depleted, the BIG-IP system reduces the traffic flow rate to the base rate of 1,000 bytes per second, with no bursting allowed.

If traffic drops back down to 800 bytes per second

No bursting is necessary, but now the BIG-IP system can add the 200 bytes per second of unused bandwidth back into the burst reservoir because the reservoir is empty. If traffic continues to flow at 800 bytes per second, the burst reservoir becomes fully replenished from 0 to 5,000 bytes in 25 seconds (at a rate of 200 bytes per second). If traffic stops flowing altogether, creating 1,000 bytes per second of unused bandwidth, then the BIG-IP system adds 1,000 bytes per second into the burst reservoir, thus replenishing the reservoir from 0 to 5,000 bytes in only 5 seconds.

About the direction setting

Using the **Direction** setting, you can apply a rate class to client or server traffic. Thus, you can apply a rate class to traffic going to a client, to a server, or to both client and server. Possible values are **Any**, **Client**, and **Server**. The default value is **Any**.

Specifying direction is useful in cases where the nature of the traffic is directionally-biased. For example, if you offer an FTP service to external clients, you might be more interested in limiting throughput for those clients uploading files to your site than you are for clients downloading files from your site. In this case, you would select **Server** as the direction for your FTP rate class, because the **Server** value only applies your throughput restriction to traffic going from the client to the server.

About the parent class

When you create a rate class, you can use the **Parent Class** setting to specify that the rate class has a parent class. This allows the child rate class to borrow unused bandwidth from the ceiling of the parent class. A child class can borrow unused bandwidth from the ceiling of its parent, but a parent class cannot borrow from a child class. Borrowing is also not possible between two child classes of the same parent class or between two unrelated rate classes.

A parent class can itself have a parent, provided that you do not create a circular dependency. A *circular dependency* is a relationship where a rate class is a child of itself, directly or indirectly.

If a rate class has a parent class, the child class can take unused bandwidth from the ceiling of the parent class. The process occurs in this way:

- If the rate of traffic flow to which the child class is applied exceeds its base rate, the child class begins to deplete its burst reservoir as described previously.
- If the reservoir is empty (or no burst size is defined for the rate class), then the BIG-IP® system takes unused base-rate bandwidth from the ceiling of the parent class and gives it to the child class.
- If the unused bandwidth from the parent class is depleted, then the child class begins to use the reservoir of the parent class.
- If the reservoir of the parent class is empty (or no burst size is defined for the parent class), then the child class attempts to borrow bandwidth from the parent of the parent class, if the parent class has a parent class.
- This process continues until there is no remaining bandwidth to borrow or there is no parent from which to borrow.

Borrowing only allows the child to extend its burst duration; the child class cannot exceed the ceiling under any circumstance.

Note: Although the above description uses the term “borrowing”, bandwidth that a child class borrows is not paid back to the parent class later, nor is unused bandwidth of a child class returned to its parent class.

About shaping policy

This setting specifies a shaping policy that includes customized values for drop policy and queue method. The default value is None.

You can create additional shaping policies using the Traffic Management shell (tmsh).

About queue method

The **Queue Method** setting determines the method and order in which the BIG-IP® system dequeues packets.

A rate class supports two queue methods:

Stochastic Fair Queue

Stochastic Fair Queueing (SFQ) is a queuing method that queues traffic under a set of many lists, choosing the specific list based on a periodically-changing hash of the connection information. This results in traffic from the same connection always being queued in the same list. SFQ then dequeues

traffic from the set of the lists in a round-robin fashion. The overall effect is that fairness of dequeuing is achieved because one high-speed connection cannot monopolize the queue at the expense of slower connections.

Priority FIFO

The *Priority FIFO (PFIFO)* queuing method queues all traffic under a set of five lists based on the Type of Service (ToS) field of the traffic. Four of the lists correspond to the four possible ToS values (Minimum delay, Maximum throughput, Maximum reliability, and Minimum cost). The fifth list represents traffic with no ToS value. The PFIFO method then processes these five lists in a way that attempts to preserve the meaning of the ToS field as much as possible. For example, a packet with the ToS field set to Minimum cost might yield dequeuing to a packet with the ToS field set to Minimum delay.

About drop policy

The BIG-IP® system drops packets whenever the specified rate limit is exceeded. A drop policy specifies the way that you want the system to drop packets. The default value is **fred**.

Note: You cannot use **fred** or **red**, if you select **sfq** for the **Queue Method** setting.

Possible values are:

fred

Specifies that the system uses Flow-based Random Early Detection to determine whether to drop packets, based on the aggressiveness of each flow. If you require flow fairness across the rate class, select **fred**.

red

Specifies that the system randomly drops packets.

tail

Specifies that the system drops the end of the traffic stream.

You can create additional drop policies using the Traffic Management shell (`tmssh`).

Using Acceleration Policies to Manage and Respond to HTTP Requests

Overview: Acceleration policies

An *acceleration policy* is a collection of defined rule parameters that dictate how the BIG-IP® system handles HTTP requests and responses. The BIG-IP system uses two types of rules to manage content: matching rules and acceleration rules. *Matching rules* are used to classify requests by object type and match the request to a specific acceleration policy. Once matched to an acceleration policy, the BIG-IP system applies the associated *acceleration rules* to manage the requests and responses.

Depending on the application specific to your site, information in requests can sometimes imply one type of response (such as a file extension of .jsp), when the actual response is a bit different (like a simple document). For this reason, the BIG-IP system applies matching rules twice: once to the request, and a second time to the response. This means that a request and a response can match to different acceleration rules, but it ensures that the response is matched to the acceleration policy that is best suited to it.

Policies screen access

The Policies screen displays all of the acceleration policies available for assignment to your applications. From the Policies screen, you can access additional screens, from which you can perform additional tasks.

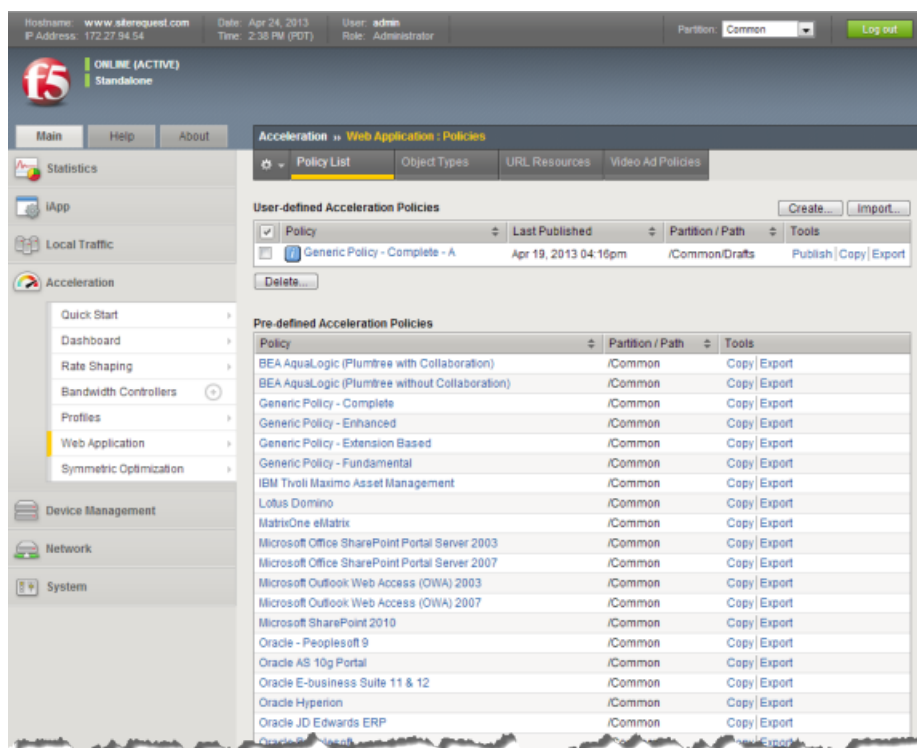


Figure 7: Example Policies screen

Types of acceleration policies

There are two types of acceleration policies that you can use to speed up the access to your web applications.

Type of policies	Description
Predefined acceleration policies	The BIG-IP ships with several predefined acceleration policies that are optimized for specific web applications, as well as four non-application specific policies for general delivery.
User-defined acceleration policies	You can create a user-defined policy by either copying an existing policy and modifying or adding rules, or by creating a new acceleration policy and specifying all new rules.

BIG-IP acceleration policies options

When configuring policies in a BIG-IP acceleration application, you can do one or more of the following tasks.

Predefined Policies

- Use a predefined policy. Predefined policies are available when you configure a BIG-IP acceleration application. You do not need to create them.

User-Defined Policies

- Create and use a user-defined policy by copying a predefined policy.
- Create and use a new user-defined policy

Acceleration policy selection

You can select a predefined acceleration policy that is associated with your specific application publisher or you can use one of the predefined generic acceleration policies. Both work well for most sites that use Java 2 Platform Enterprise Edition (J2EE) applications.

Predefined Policy	Description
Generic Policy - Complete	This predefined acceleration policy is ideal for Apache HTTP servers, Internet Information Services (IIS) web servers, WebLogic application servers, and IBM Websphere Application Servers. HTML pages are cached and Intelligent Browser Referencing is enabled.
Generic Policy - Enhanced	This predefined acceleration policy is ideal for Apache HTTP servers, Internet Information Services (IIS) web servers, WebLogic application servers, and IBM Websphere Application Servers. HTML pages are cached and Intelligent Browser Referencing is enabled for includes.
Generic Policy - Extension Based	This predefined acceleration policy is ideal for High Performance policy for Ecommerce applications that uses File Extensions instead of mime-types. This application policy is ideal if response-based matching is not required.
Generic Policy - Fundamental	This predefined acceleration policy is ideal for Apache HTTP servers, Internet Information Services (IIS) web servers, WebLogic application

Predefined Policy	Description
	servers, and IBM Websphere Application Servers. HTML pages are always proxied and Intelligent Browser Referencing is disabled.

Customization of acceleration policies

If you have a unique application for which you cannot use a predefined acceleration policy, you can create a new, user-defined acceleration policy.

Before you can create a new acceleration policy, you need to analyze the type of traffic that your site's applications receive, and decide how you want the BIG-IP to manage those HTTP requests and responses. To help you do that, consider questions similar to the following.

- Which responses do I want the BIG-IP to cache?
- Are there responses for static documents that can remain in the system's cache for several days before being refreshed?
- Which responses are dynamic documents that the BIG-IP should refresh hourly?
- Are there responses that the BIG-IP should never cache?

After you decide how you want the BIG-IP to handle certain requests for your site, you can identify the HTTP data parameters that the BIG-IP uses to match requests and responses to the appropriate acceleration policies.

For example, the path found on requests for static documents might be different than the path for dynamic documents. Or the paths might be similar, but the static documents are in PDF format and the dynamic documents are Word documents or Excel spreadsheets. These differences help you specify matching rules that prompt the BIG-IP to match the HTTP request to the acceleration policy that will handle the request and the response most expeditiously.

Creation of user-defined policies

You can create a user-defined acceleration policy most efficiently by copying an existing acceleration policy and modifying its rules to meet your unique requirements. Alternatively, you can create a new user-defined acceleration policy and define each matching rule and acceleration rule individually.

When you copy or create an acceleration policy, the BIG-IP maintains that acceleration policy as a development copy until you publish it, at which time the BIG-IP creates a production copy. Only a production (published) copy of an acceleration policy is available for you to assign to an application. You can make as many changes as you like to the development copy of an acceleration policy without affecting current traffic to your applications.

Publication of acceleration policies

When you modify rules for a user-defined acceleration policy that is currently assigned to an application, the BIG-IP creates a development copy and continues to use the currently published (production) copy to manage requests. The BIG-IP acceleration manager uses the modified acceleration policy to manage traffic only after you publish it.

If you create a new acceleration policy, you must publish it before you can assign it to an application.

About the Acceleration Policy Editor role

You can use the Acceleration Policy Editor role to manage and customize acceleration policies for the BIG-IP®. This role provides full access to acceleration features and functionality, and read-only access to all other BIG-IP features and functionality.

Acceleration policies exported to XML files

You can use the export feature to save an acceleration policy to an XML file. F5 Networks® recommends that you use the export feature every time you change a user-defined acceleration policy, so that you always have a copy of the most recent acceleration policy. You can use this file for back up and archival purposes, or to provide to the F5 Networks® Technical Support team for troubleshooting issues.

Overview: Policy Matching

The BIG-IP® system provides the flexibility needed to accelerate Web applications by processing and caching specific HTTP requests and responses. BIG-IP acceleration policies determine how the system handles and matches each request and response. A Policy Tree, configurable in the Policy Editor screen, contains branch nodes and leaf nodes that comprise a BIG-IP acceleration policy.

Leaf nodes include all of the settings (such as cache lifetime settings or proxy settings) and matching rules that determine how similar requests are processed. Additionally, grouping multiple leaf nodes under a branch node enables them to inherit the branch node settings.

When a request is received, the type of requested content typically determines the settings needed to process the request. Because `Content-Type` and `Content-Disposition` headers only become available when the BIG-IP system receives a response, the BIG-IP system provides a matching abstraction for requests called *content type* to determine, based on the request's URL and available headers, the probable or actual content type, as well as to simplify the matching rules. For example, by default, requests with an extension of `.gif` are given an object type of `images` that is used in the abstract content type, which is more convenient to use in matching rules. The mapping for each abstract content type is configured as an Identifier in the Object Types screen.

The predefined content types consist of a descriptive group name (such as `documents`) and an object type name (such as `mword`). Matching rules can require either or both parts to match, as preferred. Many of the default policies have a node for matching documents. Some use the object type abstraction and some use the URL extension.

You configure matching rules from the Matching Rules screen, and configure Acceleration Rules by choosing Acceleration Rules from the Matching Rules menu.

Matching rules for leaf nodes determine the nodes to which requests and responses apply. All matching rules for a node must match before it can be considered to be a candidate for a best match. If more than one candidate exists, resolution rules, based upon priority and precedence, determine the single best match.

Resolution rules when multiple nodes match

Sometimes, both precedence and priority can produce a match. When multiple nodes produce a match, the BIG-IP must determine the best match. In some instances, priority determines the best match, in others,

precedence determines the best match, and in still others, both precedence and priority together determine the best match.

Priority 1: An exact path match

An exact path match is one where the value set for the path parameter ends with a question mark. For example, if you have a rule with a path parameter value of `apps/srch.jsp?`, the BIG-IP considers a request of `http://www.siterequest.com/apps/srch.jsp?value=computers` to be an exact match, and matches the request to the leaf node to which the rule belongs.

It is important to note that a path of `/` and `/?` are two different things. A path that includes a `?` indicates that an exact match is required.

By default, a path that you provide for a policy is a prefix. For example, if you give a parameter the path, `/a/b`, the BIG-IP considers both of the following requests a match:

`http://www.siterequest.com/a/b?treat=bone` and
`http://www.siterequest.com/a/b/c?toy=ball`.

If you add a question mark to the parameter so that it is, `/a/b?`, the BIG-IP considers only `http://www.siterequest.com/a/b?treat=bone` to be a match, because the question mark indicates that an exact match is required.

Priority 2: A single extension node match

If no single exact path matches, but only one node matches the extension, the BIG-IP considers the request to be an exact extension match, or best match.

For example, if you have a request matching rule that specifies an extension of `jpg`, the BIG-IP considers the following request an exact extension match, and matches the request to the leaf node to which the rule belongs.

`http://www.siterequest.com/images/down.jpg`

Priority 3: A single path segment match

If no single node matches an exact path or exact extension, but only one node matches a path segment, the BIG-IP considers the request to be an exact path segment match, or best match.

Matching rules based on a path segment (the text between two slash marks) have third priority over other parameter matches. If a single path segment matches a path segment within the path, the BIG-IP matches the request to the leaf node to which the rule belongs.

For example, if you have a rule that specifies a path segment of `a`, the BIG-IP considers the following request an exact match, and matches the request to the leaf node to which the rule belongs.

`http://www.siterequest.com/a/b?treat=bone`

Priority 4: Multiple extension matches

If the request does not match a single path node, a single extension node, or a single path segment node, but multiple extension nodes match, the BIG-IP applies specific matching rules to determine the best match.

Matching rules based on multiple extension matches have a fourth priority over other parameter matches. If multiple extension matches occur, only the following rules apply.

Table 4: Multiple extension matching rules

Parameter Match	Description
One matching node with a longer path	<p>For example, if you have a rule that specifies an extension of <code>jpg</code>, the rule matches request with the longest path, specifically Node 1 of the following.</p> <p>Node 1: <code>http://www.siterequest.com/images/down.jpg</code></p> <p>Node 2: <code>http://www.siterequest.com/down.jpg</code></p>
Node that matches the most conditions, if multiple matching nodes have the same path length	<p>For example, if you have two rules that specify the same path, but the second rule also specifies a matching path segment, then the second rule matches the request to the leaf node to which the rule belongs. In this example, Node 1 in the following is matched.</p> <p>Node 1: <code>http://www.siterequest.com/images/down.jpg</code> Path Segment: <code>down (R1, 2)</code></p> <p>Node 2: <code>http://www.siterequest.com/images/down.jpg</code></p>
Node with the lowest ordinal number, if multiple matching nodes have the same path length and number of conditions	<p>For example, if you have two rules that specify the same path and include the same number of conditions, then the node with the lowest ordinal number from the policy is matched. In this example, Node 2 in the following is matched.</p> <p>Node 1: <code>http://www.siterequest.com/apps/search.jsp?dog&cat&search=magic</code> Path: <code>/apps/search.jsp</code> Path Segment: <code>cat(L2,2)</code></p> <p>Node 2: <code>http://www.siterequest.com/apps/search.jsp?dog&cat&search=magic</code> Path: <code>/apps/search.jsp</code> Path Segment: <code>dog(L2,1)</code></p>

Unmatched requests

If a request does not match a leaf node in the Policy Tree, there is an unmatched node in the Policy Tree, and the BIG-IP either uses a predefined accelerator policy that manages unmatched requests and responses, or sends the request to the origin web server for content.

It is important to keep in mind that for the BIG-IP to consider a request a match, the request must match all the matching rules configured for a leaf node. If a request matches all the rules for a leaf node, except for one, the BIG-IP does not consider it a match, and processes it as an unmatched request.

An example matching rule

This topic provides information about how to configure an example matching rule. For this example site, you have three top-level nodes on the Policy Tree.

- **Home.** This branch node specifies the rules related to the home page.
- **Applications.** This branch node specifies the rules related to the applications for the site, with the following leaf nodes.
 - **Default.** This leaf node specifies the rules related to non-search related applications.
 - **Search.** This leaf node specifies the rules related to your site's search application.
 - **Images.** This branch node specifies the rules related to graphics images.

You configure matching rules for this example Policy Tree, as described in the following table.

Table 5: Application matching rules example configuration

Node	Application matching parameter
Home	Create a rule based on the Path data type. Provide the following two values for the Path parameter. <ul style="list-style-type: none"> • /index.jsp • /?
Default	Create a rule based on the Path data type. Provide the following value for the Path parameter: /apps.
Search	Create a rule based on the Path data type. Provide the following value for the Path parameter: /srch.
Images	Create a rule based on the Path data type. Provide the following value for the Path parameter: /images.

Overview: Policy Editor screen

From the Policy Editor screen, you can view the matching rules and acceleration rules for user-defined and predefined acceleration policies, as well as create or modify user-defined acceleration policies.

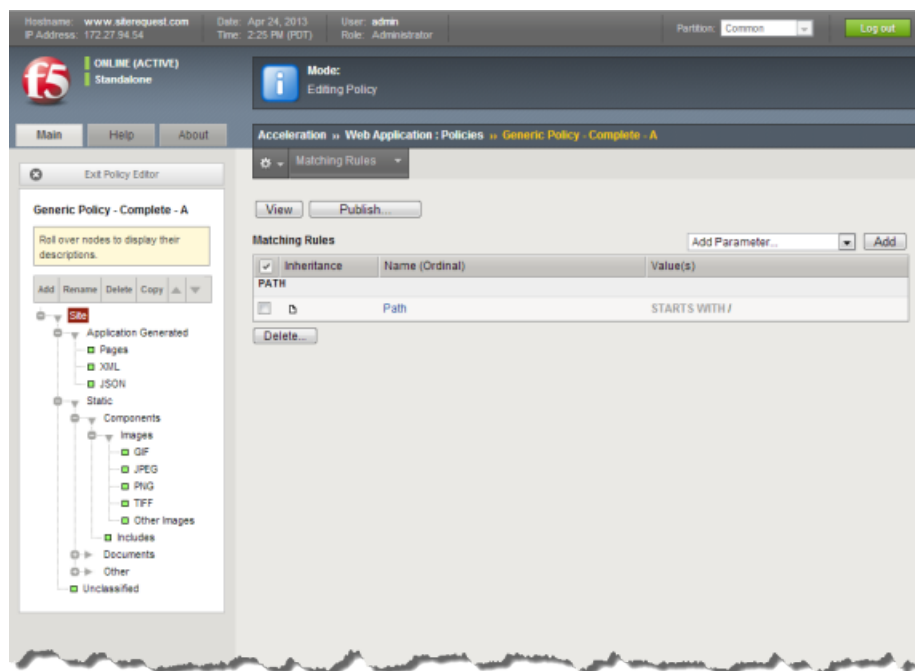


Figure 8: Policy Editor screen for an example acceleration policy

Policy Editor screen parts

There are three main parts to the Policy Editor screen.

Part	Description
Policy Tree	<p>Located on the left side of the Policy Editor screen, the Policy Tree contains branch nodes and leaf nodes, which you can modify by using the function bar. A branch node represents a group of content types (such as application generated or static) and each leaf node represents specific content (such as images, includes, PDF documents, or Word documents). The Policy Tree function bar includes the following options.</p> <ul style="list-style-type: none"> • Add. Use to create a new content type group (branch node) or a new content type (leaf node). • Rename. Use to change the name of a branch or leaf node. • Delete. Use to remove a branch or leaf node. • Copy. Use to copy a branch or leaf node. • Move Up arrow. Use to change the priority of a leaf node up within the branch node. • Move Down arrow. Use to change the priority of a leaf node down within the branch node.
Screen trail	<p>Located above the Policy Editor menu bar, the screen trail displays (horizontally) the screens that you accessed in order to arrive at the current screen. You can click the name of a screen in the trail to move back to a previous location.</p>
Policy Editor menu bar	<p>Located below the screen trail, the Policy Editor menu bar contains a list from which you select Matching Rules (default) or Acceleration Rules.</p>

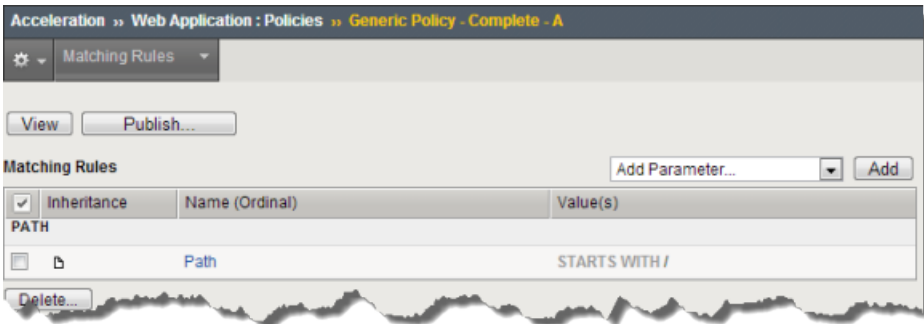


Figure 9: Matching rules displayed from the Policy Editor

When you select **Acceleration Rules**, the acceleration rules menu bar appears.



Figure 10: Policy Editor menu bar displaying acceleration rules options

Policy Tree

Matching rules and acceleration rules for acceleration policies are organized on the Policy Tree, which you access from the Policy Editor screen.

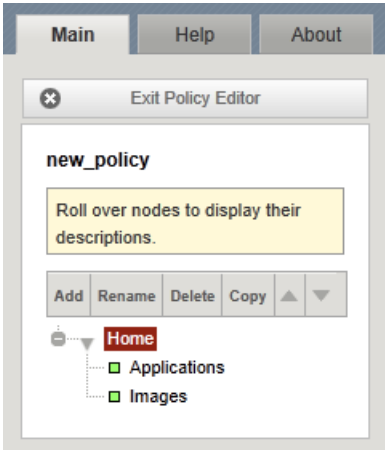


Figure 11: A Policy Tree example

Acceleration policy rule inheritance

The structure of the Policy Tree supports a parent-child relationship. This allows you to easily randomize rules. That is, because a leaf node in a Policy Tree inherits all the rules from its root node and branch node, you can quickly create multiple leaf nodes that contain the same rule parameters by creating a branch with multiple leaf nodes. If you override or create new rules at the branch node level, the BIG-IP reproduces those changes to the associated leaf nodes.

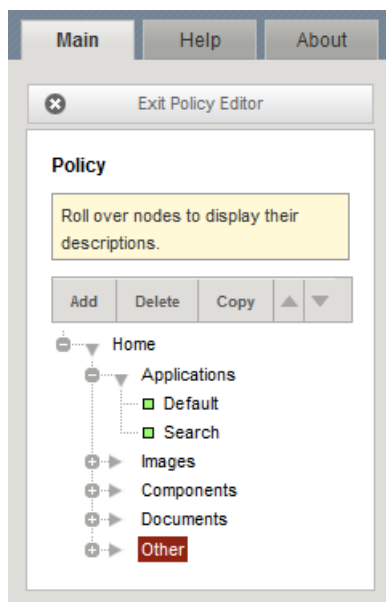


Figure 12: Rule inheritance on a Policy Tree

Nodes are defined as follows.

Node	Description
Root node	The root node exists only for the purpose of inheritance; the BIG-IP does not perform matching against root nodes. The Policy Tree typically has only one root node, from which all other nodes are created. In the example figure, the root node is Home . What distinguishes a root node from a branch node is that a root node has no parent node.
Branch node	The branch nodes exist only for the purpose of propagating rule parameters to leaf nodes; the BIG-IP does not perform matching against branch nodes. In the example figure, the branch nodes are Applications , Images , Documents , Components , and Other . Branch nodes can have multiple leaf (child) nodes, as well as child branch nodes.
Leaf node	A leaf node inherits rule parameters from its parent branch node. The BIG-IP performs matching only against leaf nodes, and then applies the leaf node's corresponding acceleration rules to the request. Leaf nodes are displayed on the Policy Tree in order of priority. If a request matches two leaf nodes equally, the BIG-IP matches to the leaf node with the highest priority. In the example figure, the leaf nodes that are displaying are Default and Search .

Inheritance rule parameters

When you create a user-defined acceleration policy by copying an existing acceleration policy, you must determine from which branch node the acceleration policy is inheriting specific rules, and decide whether you want to change the rules at the leaf node or change the rules at the branch node. To determine inheritance for a rule parameter, view the rule parameter's inheritance icon.

The following example figure illustrates matching rules for the **Path** and **Header** rule parameters for a particular leaf node.

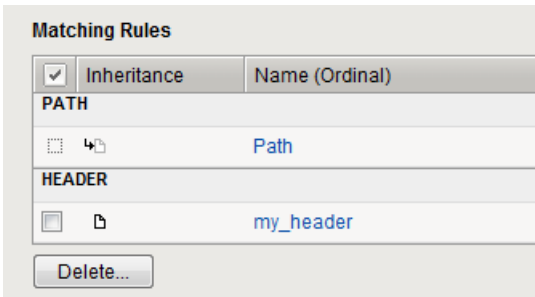


Figure 13: Inheritance example for Path and Header parameters

The arrow icon in the Inheritance column next to the **Path** parameter indicates this rule was inherited from the parent branch node. The inheritance icon next to the Header parameter does not have an arrow, indicating that the rule was not inherited; it was created locally at the leaf node.

Because the **Header** parameter rule is not inherited, you can delete the rule at the leaf node level. However, you cannot delete the **Path** parameter because it was inherited from the branch node. To delete the **Path** parameter rule, you must delete from its parent branch node.

For inherited rule parameters, you can determine the ancestor branch node by hovering the cursor over the inheritance icon. When placing the cursor on the inheritance icon next to **Path**, the branch node **Home** displays as the ancestor node, as illustrated in the following example figure.

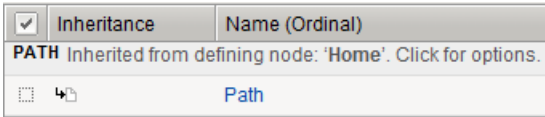


Figure 14: Inheritance example for Path parameter

Inheritance rule parameters override

When you override an inherited setting for a rule, an override icon displays (the inheritance icon with a red X) next to the rule setting. To see the node where the option was overridden, place your cursor over the override icon.

For example, for the content assembly rule in the following example figure, all of the options are inherited from the branch node, except for the **Enable Intelligent Browser Referencing To** option. For this node, the rule was disabled at the leaf node. When hovering the cursor over the override icon, a message displays next to the **Content Assembly Options** menu.

View Publish...

Content Assembly Options: Basic Overridden inheritance on node: 'Home'. Click for options.

✖ Enable Intelligent Browser Referencing To	<input type="checkbox"/>
📁 Enable Intelligent Browser Referencing Within	<input checked="" type="checkbox"/> Enabled
📁 Enable MultiConnect To	<input checked="" type="checkbox"/> Enabled
📁 Enable MultiConnect Within	<input checked="" type="checkbox"/> Enabled
📁 Enable Content Compression	<input checked="" type="checkbox"/> Enabled
📁 Enable Content Assembly on Proxies	<input checked="" type="checkbox"/> Enabled
📁 Enable Javascript and CSS Minification	<input type="checkbox"/>

Figure 15: Inheritance example with overridden rule option

To see if the current leaf node inherited this overridden option, click the parent branch node and view its rules. In the following example figure, you see that there were no rule settings overridden at the parent branch, indicating the rule was inherited from the branch node, **Home**, and overridden at the leaf node.

View Publish...

Content Assembly Options: Basic Inherited from defining node: 'Home'. Click for options.

📁 Enable Intelligent Browser Referencing To	<input checked="" type="checkbox"/> Enabled
📁 Enable Intelligent Browser Referencing Within	<input checked="" type="checkbox"/> Enabled
📁 Enable MultiConnect To	<input checked="" type="checkbox"/> Enabled
📁 Enable MultiConnect Within	<input checked="" type="checkbox"/> Enabled
📁 Enable Content Compression	<input checked="" type="checkbox"/> Enabled
📁 Enable Content Assembly on Proxies	<input checked="" type="checkbox"/> Enabled
📁 Enable Javascript and CSS Minification	<input type="checkbox"/>

Figure 16: Parent of leaf node example

When you follow this rule back to its grandparent, you see the rule options are not inherited from any other node; they are set at the grandparent node and they are all enabled, as indicated in the following example figure.

View Publish...

Content Assembly Options: Basic Local setting - Not inherited.

📁 Enable Intelligent Browser Referencing To	<input checked="" type="checkbox"/> Enabled
📁 Enable Intelligent Browser Referencing Within	<input checked="" type="checkbox"/> Enabled
📁 Enable MultiConnect To	<input checked="" type="checkbox"/> Enabled
📁 Enable MultiConnect Within	<input checked="" type="checkbox"/> Enabled
📁 Enable Content Compression	<input checked="" type="checkbox"/> Enabled
📁 Enable Content Assembly on Proxies	<input checked="" type="checkbox"/> Enabled
📁 Enable Javascript and CSS Minification	<input type="checkbox"/>

Figure 17: Grandparent of leaf node example

If you want to enable the content compression feature at the leaf node, you can use one of the following options.

- Override the inherited setting at the leaf node and select the **Enable Content Compression** check box.
- Cancel the override setting at the parent, so that the parent inherits the **Enable Content Compression** setting of the grandparent, and passes that setting to the leaf node.

Keep in mind that if you cancel the override setting at the grandparent branch node, you change the settings for all of the child leaf nodes, not just the leaf node you want to change.

Tip: *Although you have the option to override rules at the leaf node level, you should set up the Policy Tree in a logical way so that you only specify rules for branch nodes that you want all or most of its child leaf nodes to inherit. In other words, do not set a rule for a branch node if you know that most its leaf nodes will not use that rule.*

Policy Tree modification for an acceleration policy

To customize a user-defined acceleration policy, you can modify matching rules and acceleration rules for the branch and leaf nodes. Or, you can add new branch and leaf nodes and associated matching and acceleration rules to the Policy Tree.

Important: *You can edit only user-defined acceleration policies. You cannot edit predefined acceleration policies.*

Overview: HTTP header parameters

Much of the BIG-IP® device's behavior is dependent on the configured rules associated with parameters in the HTTP request headers. Although important, the presence or value of HTTP response headers does not influence as many aspects of the BIG-IP's behavior, because the BIG-IP receives HTTP response headers after performing certain types of processing.

When the BIG-IP receives a new request from a client, it first reviews the HTTP request parameters to match it to the relevant acceleration policy. After applying the associated matching rules, it sends the request to the origin web server for content.

Before sending a response to a client, the BIG-IP can optionally insert an `X-WA-Info` response header to track how it handled the request. You cannot change these informational headers, and they do not affect processing, however, they can provide useful information for evaluating your acceleration policies.

Requirements for servicing requests

To maintain high performance, the BIG-IP® does not service an HTTP request unless the request meets the following conditions.

- The request includes an HTTP request header that is no larger than 8192 bytes, and in the first line, identifies its method, URI, and protocol.
- The method for the HTTP request header is a GET, HEAD, or POST method.
- The protocol for the HTTP request header is a HTTP/0.9, HTTP/1.0, HTTP/1.1, or HTTP/2.0.
- The HTTP post data on the request is no larger than 32768 bytes.
- If the request provides the Expect request header, the value is `100-continue`.

- If the request provides the `Content-Type` request header, the value is `application/x-www-form-urlencoded`.
- The request includes a `Host` request header identifying a targeted host that is mapped to an origin server at your site.

If the HTTP `Host` request header is missing or does not have a value, the BIG-IP responds to the requesting client with a 400-series error message. If the request violates any of the other conditions, the BIG-IP redirects the request to the origin web servers for content.

About the HTTP request process

When a BIG-IP® device receives an HTTP request that meets the required conditions, the BIG-IP processes the request in accordance with this sequence.

1. The BIG-IP performs policy matching against the request and retrieves the associated acceleration rules.
2. The BIG-IP evaluates the policy matching to a proxying rule, as follows:

Condition	Process
Request is matched to a proxying rule	The BIG-IP sends the request to the origin web servers as required by the rule.
Request is not matched to a proxying rule	The BIG-IP attempts to retrieve the appropriate compiled response from cache.

3. The BIG-IP processes the attempt to retrieve the appropriate compiled response from cache, as follows:

Condition	Process
No compiled response resides in cache	The BIG-IP sends the request to the origin web servers for content.
Compiled response resides in cache	The BIG-IP searches for an associated content invalidations rule for the compiled response.

4. The BIG-IP processes the resultant content invalidations rule for the compiled response, as follows:

Condition	Process
A content invalidations rule is triggered for the compiled response	The BIG-IP compares the rule's effective time against the compiled response's last refreshed time.
A content invalidations rule is not triggered	The BIG-IP examines the compiled response's TTL value to see if the compiled response has expired.

5. The BIG-IP processes the compiled response's last refresh time, as follows:

Condition	Process
The compiled response's last refreshed time is before the content invalidations rule's triggered time	The BIG-IP sends the request to the origin web servers for content.
The compiled response's last refreshed time is after the content invalidations rule's triggered time	The BIG-IP examines the compiled response's TTL value to see if the compiled response has expired.

6. The BIG-IP processes the compiled response's TTL value, as follows:

Condition	Process
The compiled response has expired	The BIG-IP sends the request to the origin web servers.
The compiled response has not expired	The BIG-IP services the request using the cached compiled response.

Requirements for caching responses

When the BIG-IP® device receives a response from the origin web server, it inspects the HTTP response headers, applies the acceleration rules to the response, and sends the content to the client. To ensure the most effective performance, the BIG-IP does not cache a response from the origin server, or forward it to the originating requestor, unless it meets the following conditions.

- The request does not match to a do-not-cache proxying rule.
- The first line of the response identifies the protocol, a response code that is an integer value, and a response text. For example: `HTTP/1.1 200 (OK)`.
- If the `Transfer-Encoding` response header is used on the response, the value is `chunked`.
- The response is complete, based on the method and type of data contained within the response, as follows.
 - HTML tags. By default, the BIG-IP considers a response in the form of an HTML page complete only if it contains both beginning and ending HTML tags.
 - Content-Length response header. If a response is anything other than an HTML page, or if you have overridden the default behavior described in the previous bullet point, the BIG-IP considers content complete only if the response body size matches the value specified on the `Content-Length` response header.
 - Chunked transfer coding. The BIG-IP accepts chunked responses that omit the final zero-length chunk. For information about chunked transfer coding, see section 3.6 in the HTTP/1.1 specification <http://www.w3.org/Protocols/rfc2616/rfc2616-sec3.html#sec3.6>.

If the BIG-IP receives a response from the origin server that does not conform to these conditions, it does not cache the response before sending it to the client.

About the HTTP responses process

When the BIG-IP receives a response from the origin web server, the BIG-IP performs the following actions.

1. Inspects the HTTP response headers.
2. Applies the acceleration rules to the response.
3. Sends the content to the client.

Configuration of rules based on HTTP request headers

In most cases, the default values for the predefined acceleration policies are sufficient, but you can fine-tune the BIG-IP® device's behavior by creating a user-defined acceleration policy and modifying the HTTP request data type parameters. When you specify or modify an HTTP data type parameter for an acceleration

policy rule, you define specific HTTP data type parameter criteria that the BIG-IP uses to manage HTTP requests. When specifying parameter criteria, you designate the following information within a rule.

- Parameter identity. This can include one or more of the following criteria.
 - Parameter type
 - Parameter name
 - Parameter location within the HTTP request
- Parameter value or state. This can include one or more of the following parameter state and value.
 - Parameter is present in the HTTP request and matches the defined value provided in the form of a regular expression.
 - Parameter is present in the HTTP request and does not match the specified value provided in the form of a regular expression.
 - Parameter is present in the HTTP request, but has no value (is an empty string).
 - Parameter is not present in the HTTP request
- BIG-IP action. Where you specify the following criteria.
 - Whether the BIG-IP performs an action on a match or a no match.
 - The action that the BIG-IP performs, which is dictated by the rules in the associated acceleration policy.

For example, if you specify a rule that the BIG-IP performs an action when a request does not match a configured parameter, the rule triggers if the parameter in the request is a different value than you specified, or if the value is empty (null). The BIG-IP does not perform the specified action if the parameter does not appear in the request.

Specification of HTTP data type parameters for a rule

You cannot configure rules based on all HTTP data types parameters; you can only specify the parameters that the BIG-IP uses when processing HTTP requests.

Note: *Lifetime rules and responses cached rules do not use HTTP data type parameters.*

The HTTP data type parameters that the BIG-IP uses when processing HTTP requests, are defined as follows.

Note: *To specify that the parameter name is case-sensitive, enable the **Values are case sensitive** setting when configuring the parameter options.*

Host

A rule that uses the host parameter is based on the value provided for the HTTP Host request header field. This header field describes the DNS name that the HTTP request is using. For example, for the following URL the host equates to `HOST: www.siterequest.com`.

`http://www.siterequest.com/apps/srch.jsp?value=computers`

Path

A rule that uses the path parameter is based on the path portion of the URI. The path is defined as everything in the URL after the host and up to the end of the URL, or up to the question mark, (whichever comes first). The following table shows examples of URLs and paths.

Table 6: Path example

URL	Path
<code>http://www.siterequest.com/apps/srch.jsp?value=computers</code>	<code>/apps/srch.jsp</code>
<code>http://www.siterequest.com/apps/magic.jsp</code>	<code>/apps/magic.jsp</code>

Extension

A rule that uses the extension parameter is based on the value that follows the far-right period, in the far-right segment key of the URL path.

For example, in the following URLs, gif, jpg, and jsp are all extensions.

- `http://www.siterequest.com/images/up.gif`
- `http://www.siterequest.com/images/down.jpg`
- `http://www.siterequest.com/apps/psrch.jsp;SID=AAyB23?src=magic`

Query Parameter

A rule that uses the query parameter is based on a particular query parameter that you identify by name, and for which you provide a value to match against. The value is usually literal and must appear on the query parameter in the request, or a regular expression that matches the request's query parameter value. The query parameter can be in a request that uses GET, HEAD, or POST methods.

You can also create a rule that matches the identified query parameter when it is provided with an empty value, or when it is absent from the request. For example, in the following URL the action query parameter provides an empty value.

`http://www.siterequest.com/apps/srch.jsp?action=&src=magic`

Unnamed Query Parameter

An unnamed query parameter is a query parameter that has no equal sign. That is, only the query parameter value is provided in the URL of the request. For example, the following URL includes two unnamed query parameters that have the value of dog and cat.

`http://www.siterequest.com/apps/srch.jsp?dog&cat&src=magic`

A rule that uses the unnamed query parameter specifies the ordinal of the parameter, instead of a parameter name. The ordinal is the position of the unnamed query parameter in the query parameter portion of the URL. You count ordinals from left to right, starting with 1. In the previous URL, dog is in ordinal 1 and unnamed, cat is in ordinal 2 and unnamed, and src is in ordinal 3 and named magic.

**Figure 18: Query parameter: ordinal 1=dog (unnamed); ordinal 2=cat (unnamed); ordinal 3=src (named magic)**

You can create a rule that matches the identified (unnamed) query parameter when it is provided with an empty value, or when it is absent from the request. For example, in the following URL, ordinal 1 provides an empty value.

`http://www.siterequest.com/apps/srch.jsp?&cat&src=magic`

**Figure 19: Query parameter: ordinal 1=&cat (empty); ordinal 2=src (named magic)**

In the following URL, ordinal 3 is absent (dog is in ordinal 1 and src is in ordinal 2).

`http://www.siterequest.com/apps/srch.jsp?dog&src=magic`



Figure 20: Query parameter: ordinal 1=dog (empty); ordinal 2=src (named magic); ordinal 3 (absent)

Path Segment

A rule that uses the path segment parameter identifies one of the following values.

- Segment key
- Segment parameter

Segment key. A segment is the portion of a URI path that is delimited by a forward slash (/). For example, in the path: `/apps/search/full/complex.jsp`, `apps`, `search`, `full`, and `complex.jsp` all represent path segments. Further, each of these values are also the segment key, or the name of the segment.

Segment parameter. A segment parameter is the value in a URL path that appears after the segment key. Segment parameters are delimited by semicolons. For example, `magic`, `shop`, and `act` are all segment parameters for their respective path segments in the following path.

`/apps/search/full;magic/complex.jsp;shop;act`

To specify segment parameters, you must also identify segment ordinals.

Segment ordinal. To specify a segment for a rule, you must provide an ordinal that identifies the location of the segment in the following path.

`/apps/search/full;magic/complex.jsp;shop;act`

You must also indicate in the rule, which way you are counting ordinals in the path: from the left or the right (you always count starting at 1). For the example shown, `/full;magic`, the ordinals for this path are as show in the following table.

Table 7: Segment ordinals example

Ordinal Numbering Selection	
3	Numbering Left-to-Right in the Full Path
2	Numbering Right-to-Left in the Full Path

Cookie

A rule that uses the cookie parameter is based on a particular cookie that you identify by name, and for which you provide a value to match against. Usually the value is literal and must appear on the cookie in the request, or a regular expression that must match the request's cookie that appears on the cookie HTTP request headers. These are the same names you use to set the cookies, using the `HTTP SET-COOKIE` response headers.

You can also create a rule that matches when the identified cookie is provided with an empty string or when it is absent from the request. For example, in the following string, the following `REPEAT` cookie is empty.

`COOKIE: REPEAT=`

In the following string, the `USER` cookie is present and the `REPEAT` cookie is absent.

`COOKIE: USER=334A5E4`

User Agent

A rule that uses the user agent parameter is based on the value provided for the `HTTP USER_AGENT` in the request header, which identifies the browser that sent the request. For example, the following `USER_AGENT` request header indicates that the requesting browser is IE 5.01 running on Windows NT 5.0.

```
USER_AGENT: Mozilla/4.0 (compatible; MSIE 5.01; Windows NT 5.0)
```

You do not typically base rules on the `USER_AGENT` request header, unless your site behaves differently depending on the browser in use.

Referrer

A rule that uses the referrer parameter is based on the value provided for the `HTTP REFERER` in the request header. (Note the misspelling of `REFERER`. This spelling is defined for this request header in all versions of the HTTP specification.)

This header provides the URL location that referred the client to the page that the client is requesting. That is, `REFERER` provides the URL that contains the hyperlink that the user clicked to request the page. For example, the following `REFERER` request header provides the referred URL of

```
http://www.siterequest.com/.
```

```
REFERER: http://www.siterequest.com/
```

You do not typically base rules on the `REFERER` request header, unless you want your site's behavior to be dependent on the specific referrer. For example, one implementation would be for sites that provide different branding for their pages based on the user's web portal or search engine.

Protocol

A rule that uses the protocol parameter is based on whether the request uses the HTTP or HTTPS protocol. For example, the following URL uses the HTTP protocol.

```
http://www.siterequest.com/apps/srch.jsp?value=computers
```

The following URL uses the HTTPS protocol.

```
https://www.siterequest.com/apps/srch.jsp?value=computers
```

Method

A rule that uses the method parameter is based on whether the request used the `GET`, `HEAD`, or `POST` method.

Header

A rule that uses the header parameter is based on a particular header that you identify by name, and for which you provide a value to match against. You can use an HTTP request data type header parameter to create rules based on any request header, other than one of the recognized HTTP request data types.

The HTTP request data type header parameter you use can be standard HTTP request header fields such as `AUTHORIZATION`, `CACHE-CONTROL`, and `FROM`. They can also be user or acceleration defined headers in the form of a structured parameter.

Following are examples of HTTP request data type parameters.

- `Accept: text/html, image/*`
- `Accept-Encoding: gzip`
- `Accept-Language: en-us`
- `CSP-Gadget-Realm-User-Pref: NM=5,PD=true,R=3326`

The last header in the example depicts a structured parameter.

The format of a structured parameter in a request is similar to that used for a cookie, with a header name that you choose, followed by a series of `name=value` pairs separated by commas. The header name is not case-sensitive and in this structure, the semicolons (;) are special characters. The parser ignores anything

after a semicolon until it reaches the subsequent comma. For example, following are valid header structured parameters.

- `CSP-Global-Gadget-Pref: AT=0`
- `CSP-Gadget-Realm-User-Pref: NM=5,PD=true,R=3326`
- `CSP-User-Pref:`
- `E2KU=chi,E2KD=ops%u002esiterequest%u002enet,E2KM=chi`
- `CSP-Gateway-Specific-Config:`
- `PT-User-Name=chi,PT-User-ID=212,PT-Class-ID=43`
- `Standards: type=SOAP;SOAP-ENV:mustUnderstand="1",version=1.2`

In the last line, the parser ignores `SOAP-ENV:mustUnderstand="1"`, because it follows a semicolon. Since `version=1.2` follows the command, the parser reads it as a `name=value` pair. If you have metadata that you want to include in the header, but want the BIG-IP to ignore, put it after a semicolon.

If you specify a header as a structured parameter when creating a rule, the BIG-IP module parses it into `name=value` pairs when it examines the request. If you do not specify it as a structured parameter, the BIG-IP processes it like a normal header, and treats everything after the colon (:) as the value. To define a header as a structured parameter when you are creating or editing a rule, you specify the name using the following syntax: `headername:parmname`, where `headername` is the name of the header and `parmname` is the name of the parameter with the value that you want to affect the rule.

Using the `CSP-Global-Gadget-Pref` header as an example, if you want the BIG-IP to evaluate the value for `AT` to determine if a configured rule should apply, specify the name of the header parameter as follows: `CSP-Global-Gadget-Pref:AT`.

If you want the BIG-IP to evaluate the entire string without assigning meaning to `name=value` pairs, specify the name of the header parameter as follows: `CSP-Global-Gadget-Pref`.

You can create a rule that matches when the identified header is provided with an empty value, or when it is absent from the request.

In the following example, the BIG-IP considers `Standards:release`, empty and considers `Standards:SOAP-ENV` absent, because it is ignored: `Standards: type=SOAP;SOAP-ENV:mustUnderstand="1",release=,version=1.2`.

Client IP

A rule that uses the client IP parameter is based on the IP address of the client making the request. The IP address, however, might not always be the address of the client that originated the request.

For example, if the client goes through a proxy server, the IP address is the IP address of the proxy server, rather than the client IP address that originated the request. If several clients use a specific proxy server, they all appear to come from the same IP address.

Content Type

Unlike the HTTP request data types, a matching rule based on content type is specific to the content type parameter that the BIG-IP generates for a response. You specify the regular expression that you want a response's content type to match.

Configuration of rules based on HTTP response headers

After the BIG-IP[®] receives a response from the origin web server, it performs the following processes.

- Classifies the response
- Applies associated acceleration policy rules
- Assembles the response

Response headers have no effect on application matching, variation, or invalidations rules. The BIG-IP evaluates response headers associated with caching after it compiles, but before it caches, the response. Once the BIG-IP begins the compilation and assembly process, it then examines existing response headers that influence assembly.

You can configure assembly, proxying, lifetime, or responses cached rules based on response headers.

Note: *If you configure proxying rules based on HTTP response header parameters, you can use them only in terms of how the BIG-IP caches the responses, because the BIG-IP has already sent the request to the origin web servers when it reviews the response headers.*

Classification of responses

After the BIG-IP® device receives a response from the origin server, and before it performs application matching, it classifies the response based on the object types that are defined on the Object Types screen. The BIG-IP bases this classification on the first item it finds, in the following order.

1. The file extension in the file name field of the response's `Content-Disposition` header.
2. The file extension in the extension field of the response's `Content-Disposition` header.
3. The response's `Content-Type` header, unless it is an ambiguous MIME type.
4. The request's path extension.

For example, if the extension in the file name field of the response's `Content-Disposition` header is empty, the BIG-IP looks at the response's `Content-Disposition` header's file extension in the extension field. If that has an extension, the BIG-IP attempts to match it to a defined object type. If there is no match, the BIG-IP assigns an object type of `other` and uses the settings for `other`. The BIG-IP examines the information in the `Content-Type` header only if there is no extension in the file name or extension fields of the `Content-Disposition` header.

If the BIG-IP finds a match to an object type, it classifies the response with that object type, group, and category, and uses the associated settings for compression. The object type and group under which the response is classified is also included in the `X-WA-Info` response header.

Important: *If you have defined a compression setting for an object from the Object Types screen, it overrides any compression setting configured for an assembly rule for the response's matched node.*

Once it classifies the response by object type, the BIG-IP appends it as follows: `group.objectType`. The BIG-IP then matches the response to a node of a Policy Tree in an acceleration policy (the first matching process was for the request), using the new content type. In many cases, this content type is the same as the content type for the request, and the BIG-IP matches the response to the same node as the request.

Note: *The BIG-IP also matches the response to the same node as the request if there is no `Content Type` parameter specified in a matching rule.*

Unlike the HTTP request data types, you do not base a matching rule directly on the value of an HTTP response data type. Instead, you base the rule on the content type parameter that the BIG-IP generates, by specifying the regular expression that you want a request or response's content type to match, or not to match.

Application of association acceleration policy rules

The BIG-IP® device compiles the response, and determines if it can cache it by looking for a responses cached rule for the node that matches the response. If the response is cacheable, the BIG-IP caches a copy of the response.

Assembly of responses

The BIG-IP® device assembles responses using the following information.

- The assembly rules specified for the node that matches the response.
- The **Enable Compression** setting (configured from the Object Types screen) for the object type under which the response was classified. The options for this setting include the following.
 - **Policy Controlled.** The BIG-IP uses the compression settings configured in the acceleration policy's assembly rules.
 - **None.** The BIG-IP never compresses the response. This option overrides the compression setting in the acceleration policy's assembly rules. Select this option only if you want the BIG-IP to ignore assembly rules for the specified object type.

Regular expressions and meta tags for rules

When the BIG-IP® performs pattern matching based on regular expressions, it assumes all regular expressions are in the form of `^expression$`, even if you do not explicitly set the beginning of line (^) and end of line (\$) indicators. For substring searches, you enter `*expression.*` as a regular expression.

The string that the BIG-IP matches is dependent on the HTTP data type for which you are providing the regular expression. Before the BIG-IP attempts to match information in an HTTP request to the HTTP request data type parameters, it translates any escaped characters (such as %2F or %2E) back to their regular form, such as (/ or .).

Management of Cache-Control response headers

The HTTP Cache-Control version 1.1 header specification identifies general-header field directives for all caching mechanisms along the request/response chain, and is used to prevent caching behavior from adversely interfering with the request or the response. (For additional information, see sections 13 and 14 of the HTTP/1.1 specification at <http://www.w3.org/Protocols/rfc2616/rfc2616.html>.)

Directives can appear in response or request headers, and certain directives can appear in either type of header. The HTTP Cache-Control general-header field directives typically override any default caching algorithms.

The origin web server's cache response directives are organized in two groups: `no-cache` and `max-age`.

Cache-Control: no-cache directives

By default, the majority of the BIG-IP's acceleration policies are configured to cache responses and ignore the following HTTP Cache-Control header's `no-cache` directives.

- `Pragma: no-cache`
- `Cache-Control: no-cache`
- `Cache-Control: no-store`
- `Cache-Control: private`

You can configure the BIG-IP to honor `no-cache` directives. However, doing so can result in a noticeable increase in the traffic sent to the origin web server, depending on how many users send `no-cache` directives in requests.

Cache-Control: max-age directives

The BIG-IP® system uses the HTTP Cache-Control header's max-age or s-maxage directives to determine the TTL values for compiled responses, only when the **Honor Headers From Origin Web Server** check box is selected. It uses combinations of the **max_age** and **s_maxage** settings for **Origin Web Server Headers** as selected (either one or both). If the **Honor Headers From Origin Web Server** settings are not configured, then the BIG-IP system uses the WebAccelerator Cache Settings Maximum Age value.

***Note:** If the TTL values for the s-maxage and the max-age directives are different, the BIG-IP system uses the TTL value for the s-maxage directive.*

X-WA-Info response headers

Before sending a response to a client, the BIG-IP can optionally insert an X-WA-Info response header that includes specific codes describing the properties and history of the object. The X-WA-Info response header is for informational purposes only and provides a way for you to assess the effectiveness of your acceleration policy rules.

Following is an example of an X-WA-Info header.

```
X-WA-Info: [V2.S11101.A13925.P76511.N13710.RN0.U3756337437].[OT/images.OG/images]
```

The code is divided into fields by the period (.) delimiter. Each field begins with a letter code, followed by one or more letters or numbers. The object type and group under which the response is classified are also included in the X-WA-Info response header.

The object type is preceded by OT and the group is preceded by OG, as in the following example.

```
[OT/msword.OG/documents]
```

When you enable the **X-WA-Info Header** setting for an application, the following tasks describe how to view X-WA-Info response headers.

- Perform a packet capture of the page.
- Using an HTTP viewer utility like HttpWatch, HTTP Analyzer, or Live Headers.
- Using the **Request Logging** profile.

X-WA-Info response header in a symmetric deployment example

Typical X-WA-Info header in a symmetric deployment

The X-WA-Info response header in a symmetric deployment includes keywords that designate Central and Remote devices, which can be used, as necessary, for analysis. This example shows a typical response header, and the sequence of the X-WA-Info response header in a symmetric deployment.

```
X-WA-Info: [Central].[V2.S10201.A53316.P67996.N37551.RN0.U3955110987].[OT/jpeg.OG/images].[P/0.4].[O/0.3].[EH4/19].[DH3/2].[C/D]
[Remote].[V2.S11101.A53316.P67996.N37551.RN0.U3955110987].[OT/jpeg.OG/images].[P/0.0].[O/0.3].[EH2/5].[DH2/19].[C/D]
```

The Central device's X-WA-Info header is served by the Remote device, and is only updated if one the following conditions is met.

Condition	Description
Condition 1	<ul style="list-style-type: none"> A 200 OK response is delivered from the origin web server The content has expired The content is different from the cached object on both devices
Condition 2	The cache is cleared on the Remote device but not on the Central device, causing the Central device to issue a 200 OK response to the Remote device

Sequence of X-WA-Info response header for symmetric devices

The following sequence shows the way in which the X-WA-Info response header updates in a symmetric deployment, starting with a request for new content on an empty cache with a lifetime of 2 seconds.

A first request is initiated for creation of a positive cache entry on both devices (a 200 OK response from the origin web server).

```
X-WA-Info: [Central].[V2.S10201.A53316.P67996.N37551.RN0.U3955110987].
[OT/jpeg.OG/images].[P/0.0].[O/0.3].[EH5/34].[DH1/0].[C/P]
[Remote].[V2.S10201.A53316.P67996.N37551.RN0.U3955110987].
[OT/jpeg.OG/images].[P/0.2].[O/0.3].[EH1/0].[DH1/0].[C/P]
```

A second request is initiated for caching of content on both devices (again, a 200 OK response from the origin web server).

```
X-WA-Info: [Central].[V2.S10201.A53316.P67996.N37551.RN0.U3955110987].
[OT/jpeg.OG/images].[P/0.1].[O/0.3].[EH6/14].[DH3/3].[C/D]
[Remote].[V2.S10201.A53316.P67996.N37551.RN0.U3955110987].
[OT/jpeg.OG/images].[P/0.3].[O/0.3].[EH3/1].[DH3/1].[C/D]
```

A third request is initiated once the objects are in the cache on both devices. The content is served from the Remote device's cache, and content is not proxied to the origin web server.

```
X-WA-Info: [Central].[V2.S10201.A53316.P67996.N37551.RN0.U3955110987].
[OT/jpeg.OG/images].[P/0.1].[O/0.3].[EH6/14].[DH3/3].[C/D]
[Remote].[V2.S11101.A53316.P67996.N37551.RN0.U3955110987].
[OT/jpeg.OG/images].[P/0.4].[O/0.3].[EH2/68].[DH1/2].[C/D]
```

Observe that the Central device's X-WA-Info header continues to use an S code of S10201.

A fourth request is initiated for the content that is now expired. The content is expired on both devices, and a conditional GET request is sent to the origin web server.

```
X-WA-Info: [Central].[V2.S10201.A53316.P67996.N37551.RN0.U3955110987].
[OT/jpeg.OG/images].[P/0.1].[O/0.3].[EH8/14].[DH3/3].[C/D]
[Remote].[V2.S10232.A53316.P67996.N37551.RN0.U3955110987].
[OT/jpeg.OG/images].[P/0.4].[O/0.3].[EH2/71].[DH1/6].[C/D]
```

Again, the Central device's X-WA-Info header uses an S code of S10201. The Central device is showing the S code from the last 200 OK response delivered to the Remote device by the Central device.

V code

The V code is the first field of the X-WA-Info response header. This field code indicates the version of the X-WA-Info response header code used in the BIG-IP® software.

V code	Description
V2	Version 2 of the X-WA-Info response header code, used in BIG-IP version 11.3 software.

S code

The S code is the second field of the X-WA-Info response header. This field code indicates whether the object in the HTTP response was served from the system's cache, or was sent to the original web server for content.

A code

The A code is the third field of the X-WA-Info response header, and identifies to which application the BIG-IP matched the request. This helps you determine which acceleration rules the BIG-IP applied to the request.

P code

The P code is the fourth field of the X-WA-Info response header, and it indicates the acceleration policy that the BIG-IP applied to the request.

N code

The N code is the fifth field of the X-WA-Info response header, and it identifies the application match of a request to an acceleration policy. The request node ID matches the policy node ID.

RN code

The RN code is the sixth field of the X-WA-Info response header, and it identifies the application match of a response to an acceleration policy. The BIG-IP can perform response-based application matching against MIME types in a response, or by matching attachment file name extensions. Request-based application matching against extensions in the URL path is not considered a response application match; however, a request application match appears in the N field.

A 0 (zero) RN code in the X-WA-Info response header indicates that the BIG-IP response matching did not override the decision made in the initial request match (N-code), and that the initial request match and the response match are the same. A nonzero RN code in the X-WA-Info response header indicates that the response matching overrode the decision made in the initial request match, and that the initial request match and the response match are different.

U code

The U code is the seventh field of the X-WA-Info response header and is used to verify the behavior of variation rules. Two responses with the same U code value were served the same cache entry; two responses with different U code values were served different cache entries.

Reference summary for HTTP data

This section provides HTTP reference data, including request data type parameters, response status codes, S code definitions.

HTTP request data type parameters

This table describes the HTTP request data type parameters and respective rules.

Parameter	Matching Rules	Variation Rules	Assembly Rules	Proxying Rules	Invalidations Rules
Host	x	x		x	x
Path	x				x
Extension	x				x
Query parameter	x	x	x	x	x
Unnamed query parameter	x	x	x	x	x
Path segment	x	x	x	x	x
Cookie	x	x		x	x
User Agent	x	x		x	x
Referrer	x	x		x	x
Protocol	x	x		x	x
Method	x	x		x	x
Header	x	x		x	x
Client IP	x	x		x	x
Content Type	x				

Response status codes

This table describes HTTP response codes that you can add in addition to the default 200, 201, 203, or 207 response codes.

Response code	Definition
200	OK. The request is satisfactory.
201	Created. The requested resource (server object) was created.
203	Non-Authoritative Information. The transaction was satisfactory; however, the information in the entity headers came from a copy of the resource, instead of an origin web server.
207	Multi-Status (WebDAV). The subsequent XML message might contain separate response codes, depending on the number of sub-requests.

Response code	Definition
300	Multiple Choices. The requested resource has multiple possibilities, each with different locations.
301	Moved Permanently. The requested content has been permanently assigned a new URI. The origin web server is responding with a redirect to the new location for the content.
302	Found. The requested content temporarily resides under a different URI. The redirect to the new location might change.
307	Temporary Redirect. The requested content temporarily resides under a different URI. The redirect to the new location might change.
410	Gone. The requested content is no longer available and a redirect is not available.

S code definitions

This table describes the S codes, the first field of the X-WA-Info response header, which indicates whether the object in the HTTP response was served from the system's cache, or was sent to the original web server for content.

Code	Definition	Description
SO	Response was served from an unknown source.	Indicates that the BIG-IP was unable to determine if a response was served from cache or sent to the origin web server for content.
S10101	Response was served from cache.	Indicates that the content was served from cache, that the content is usually dynamic, and that the content might or might not be assembly processed.
S10201	Response was served from the origin web server, because the request was for new content.	When the BIG-IP receives a request for new content, it sends the request to the origin web server and caches the content before responding to the request. Future requests for this content are served from cache.
S10202	Response was served from the origin web server, because the cached content had expired.	When the BIG-IP receives a request for cached content that exceeds a lifetime rule's Maximum Age setting, it revalidates the content with the origin web server (which responds with a 200 (OK) status code). After revalidating the content, the BIG-IP serves requests from cache, until the Maximum Age setting is once again exceeded.
S10203	Response was served from the origin web server, as dictated by an acceleration policy rule.	When the BIG-IP matches a request to a node with a proxying rule set to Always proxy requests for this node , the BIG-IP sends that request to the origin web server, rather than serving content from cache.
S10204	Response was served from the origin web server, because of specific HTTP or web service methods.	If the BIG-IP receives a request that contains an HTTP <code>no-cache</code> directive, the BIG-IP sends the request to the origin server and does not cache the response. In addition, the BIG-IP does not currently support some vendor-specific HTTP methods (such as OPTIONS) and some web services methods (such as SOAP). The BIG-IP sends requests containing those methods to the origin web server for content.

Code	Definition	Description
S10205	Response was served from the origin web server because the cached content was invalidated.	You can perform a cache invalidation manually, through direct XML messaging over HTTPS on port 8443, or with an acceleration rule setting. After the BIG-IP invalidates cache and retrieves new content from the origin web server, it stores the response and serves future requests from cache.
S10206	Response was served from the origin web server, because the content cannot be cached.	If a response cannot be cached, for example, because the content is private or the header is marked as <code>no-store</code> , the BIG-IP includes this code in the response to the client.
S10232	Response was served from cache, after sending a conditional <code>GET</code> request to the origin web server and receiving a response indicating that the expired cached content is still valid.	If an acceleration rule prompts the BIG-IP to expire content, it sends the next request to the origin web server. If the origin web server indicates that the cached content has not changed (responding with a <code>304</code> (Not Modified) status code), the BIG-IP includes this code in the response to the client.
S10413	Response bypassed the BIG-IP.	When a request includes an <code>Expect: 100-continue</code> header, that request and its response bypass the BIG-IP, which responds with an <code>X-WA-Info</code> header that includes an S10413 field code.
S11101	Response was served from cache.	Indicates that the content was served from cache, that the content is static, and that the content came directly from cache without assembly processing by the BIG-IP system. This is the most efficient and fastest response method.

HTTP data types for regular expression strings

This table describes the HTTP data types that are supported by the BIG-IP for regular expression strings.

HTTP data type	Definition	Example
host	The value set for the HTTP Host request header	<p>HOST: <code>www.siterequest.com</code></p> <p>The BIG-IP matches the example HTTP Host request header to the string <code>www.siterequest.com</code>.</p>
path	The value set for the path portion of the URI	<p><code>http://www.siterequest.com/apps/search.jsp?value=computer</code></p> <p>For the example URI, the BIG-IP matches the string <code>/apps/search</code>.</p>
extension	The value set for the extension portion of the URI	<p><code>http://www.siterequest.com/apps/search.jsp?value=computer</code></p> <p>For the example URI, the BIG-IP matches the string <code>jsp</code>.</p>
query parameter	The value set for the identified query parameter	<p><code>http://www.siterequest.com?action=display&PDA</code></p> <p>The BIG-IP matches the example value set for the <code>action</code> query parameter to the string <code>display</code>.</p>

HTTP data type	Definition	Example
unnamed query parameter	The value set for the identified query parameter	<p>A query parameter is matched against the requested URL, or a Content-Type header's URL encoded string in the body of a POST method. If the specified query parameter appears one or more times in the request, all instances will be matched.</p> <p><code>http://www.siterequest.com?action=display&PDA</code></p> <p>If the value set for the unnamed query parameter in ordinal 2 is this URI, the BIG-IP matches the string PDA.</p> <p>An unnamed query parameter is matched against the requested URL, or a Content-Type header's URL encoded string in the body of a POST method. The ordinal specifies the left-to-right position of the parameter to match.</p>
path segment	The name of the segment key or the value set for the segment parameter, depending on what you identify for the match	<p><code>http://www.siterequest.com/apps;AAY34/search.jsp?value=computer</code></p> <p>If you identify the path segment in ordinal 1 for the full path (counted from left-to-right), then you have identified the segment key in the URL, and the BIG-IP matches the string apps.</p> <p>Path segments are matched against the ordered segments, separated by a virgule (/) and terminated by the first question mark (?) or octothorpe (#) in the URL</p>
cookie	The value set for the identified cookie	<p><code>COOKIE: SESSIONID=TN2MM1QQL</code></p> <p>The BIG-IP matches the string TN2MM1QQL.</p>
user agent	The value set for the HTTP USER_AGENT request header	<p><code>USER_AGENT: Mozilla/4.0 (compatible; MSIE 5.5; Windows NT 5.0)</code></p> <p>The BIG-IP matches the string Mozilla/4.0 (compatible; MSIE 5.5; Windows NT 5.0).</p>
referrer	The value set for the HTTP REFERER request header	<p><code>REFERER: http://www.siterequest.com?action=display</code></p> <p>The BIG-IP matches the string <code>http://www.siterequest.com?action=display</code>.</p>
protocol	The protocol used for the request	<p><code>http://www.siterequest.com</code></p> <p>The BIG-IP matches the string http.</p>
method	The value set for the identified method	<p><code>GET /www/siterequest/index.html</code></p> <p>The BIG-IP matches the string <code>/www/siterequest/index.html</code>.</p>
header	The value set for the identified header	<p><code>Accept-Encoding: gzip</code></p> <p>The BIG-IP matches the string gzip.</p>
client ip	The source IP address for the HTTP request	<p><code>CLIENT-IP: 192.160.10.3</code></p> <p>The BIG-IP matches the string 192.160.10.3.</p>

Max age value for compiled responses

This table describes the max age values for compiled responses.

Priority	Directive	Description
1	Cache-Control: s-maxage	The BIG-IP bases the TTL on the current time, plus the value specified for the HTTP Cache-Control header's s-maxage directive. Values for this directive are expressed in seconds.
2	Cache-Control: max-age	The BIG-IP bases the TTL on the current time, plus the value specified for the HTTP Cache-Control header's max-age directive. Values for this directive are expressed in seconds.
3	Expires	The BIG-IP uses the TTL provided for HTTP Cache-Control header's entity-header field. Values for this field are expressed in Coordinated Universal Time (UTC time). To avoid caching issues, the BIG-IP must be properly synchronized with the origin web server. For this reason, F5 Networks® recommends that you configure a Network Time Protocol (NTP) server.
4	Last-Modified	<p>The BIG-IP bases this TTL by using the formula $TTL = curr_time + (curr_time - last_mod_time) * last_mod_factor$.</p> <p>In this formula, the variables are defined as follows:</p> <ul style="list-style-type: none"> <code>curr_time</code> is the time that the response is received by the BIG-IP. <code>last_mod_time</code> is the time specified by this directive. <code>last_mod_factor</code> is a percentage value used to weight the TTL you set it with a lifetime rule.

Meta characters

This table describes the meta characters that are supported by the BIG-IP for pattern matching.

Meta character	Description	Example
.	Matches any single character.	
^	Matches the beginning of the line in a regular expression. The BIG-IP assumes that the beginning and end of line meta characters exist for every regular expression it sees.	
\$	Matches the end of the line. The BIG-IP assumes that the beginning and end of line meta characters exist for every regular expression it sees.	<p>The expression <code>G.*P.*</code> matches:</p> <ul style="list-style-type: none"> GrandPlan GreenPeace GParse GP <p>A pattern starting with the <code>*</code> character is the same as using <code>.*</code> For example, the BIG-IP</p>

Meta character	Description	Example
		<p>interprets the following two expressions as identical.</p> <ul style="list-style-type: none"> • <code>*Plan</code> • <code>. *Plan</code>
<code>*</code>	Matches zero or more of the patterns that precede it.	
<code>+</code>	Matches one or more of the patterns that precede it.	<p>The expression <code>G.+P.*</code> matches:</p> <ul style="list-style-type: none"> • <code>GrandPlan</code> • <code>GreenPeace</code> <p>Do not begin a pattern with the <code>+</code> character. For example, do not use <code>+Plan</code>. Instead, use <code>.+Plan</code>.</p>
<code>?</code>	Matches none, or one of the patterns that precede it.	<p>The expression <code>G.?P.*</code> matches:</p> <ul style="list-style-type: none"> • <code>GParse</code> • <code>GP</code> <p>Do not begin a pattern with the <code>?</code> character. For example, do not use <code>?Plan</code>. Instead, use <code>.?Plan</code>.</p>
<code>[...]</code>	Matches a set of characters. You can list the characters in the set using a string made of the characters to match.	<p>The expression <code>C[AHR]</code> matches:</p> <ul style="list-style-type: none"> • <code>CAT</code> • <code>CHARISMA</code> • <code>CRY</code> <p>You can also provide a range of characters by using a dash. For example, the expression <code>AA[0-9]+</code> matches:</p> <ul style="list-style-type: none"> • <code>AA269812209</code> • <code>AA2</code> <p>It does not, however, match <code>AAB2</code>.</p> <p>To match any alphanumeric character, both upper-case and lower-case, use the expression <code>[a-zA-Z0-9]</code>.</p>
<code>[^...]</code>	Matches any character not in the set. Just as with the character, <code>[...]</code> , you can specify the individual characters, or a range of characters by using a dash (<code>-</code>).	<p>The expression <code>C[^AHR].*</code> matches:</p> <ul style="list-style-type: none"> • <code>CLEAR</code> • <code>CORY</code> • <code>CURRENT</code> <p>The expression <code>C[^AHR].*</code>, however, does not match:</p> <ul style="list-style-type: none"> • <code>CAT</code> • <code>CHARISMA</code> • <code>CRY</code>

Meta character	Description	Example
(...)	Matches the regular expression contained inside the parenthesis, as a group.	The expression AA(12)+CV matches: <ul style="list-style-type: none"> • AA12CV • AA121212CV
exp1 exp2	Matches either exp1 or exp2, where exp1 and exp2 are regular expressions.	The expression AA([de]12 [zy]13)CV matches: <ul style="list-style-type: none"> • AAd12CV • AAe12CV • AAz12CV • AAy13CV

Advanced Debug settings for General Options

For the **General Options** list, this table describes **Advanced** controls for **Debug Options**.

Advanced control	Default	Description
X-WA-Info Header	None	<p>This setting is used for troubleshooting purposes. You should not change this setting unless instructed to do so by an F5 Network Technical Support Engineer.</p> <ul style="list-style-type: none"> • None. Disables Debug Options functionality. • Standard. Enables the BIG-IP to insert an <code>X-WA-Info</code> response header, which includes specific codes that describe the properties and history of objects. • Debug. Includes advanced troubleshooting parameters.

Differentiating Requests and Responses with Variation Rules

Overview: Variation rules

When the BIG-IP® system caches responses from the origin web server, it uses certain HTTP request parameters to create a Unique Content Identifier (UCI). The BIG-IP system stores the UCI in the form of a compiled response, and uses the UCI to easily match future requests to the correct content in the module's cache.

You can configure variation rules to add or modify the parameters on which the BIG-IP system bases its caching process. If the BIG-IP system receives two requests that are identical except for the value of a query parameter defined in the variation rule, it creates a different UCI for each, and caches each response under its unique UCI.

Consider a site that receives requests from customers and partners, and wants to serve different content to each. For this site, you could create a variation rule in which you specify that when a request contains a `version` cookie set to a value of 1, the BIG-IP system serves a page specifically for customers, and when the `version` cookie is set to a value of 2, it serves a page specifically for partners. For this rule, the BIG-IP system caches the following three compiled responses.

- For content produced for `Cookie: version=1`.
- For content produced for `Cookie: version=2`.
- For content produced when the `version` cookie does not appear in the request.

Note: When configuring this variation rule, you must specify a value for the `version` cookie parameter. If you do not, the BIG-IP system ignores the cookie's value and produces, at most, two compiled responses: one for requests that contain the cookie, and one for requests that do not contain the cookie. The BIG-IP system then serves the first response it caches to any subsequent requests that contain that cookie.

Cache efficiency improvement

By default, the BIG-IP includes the following HTTP request parameters in the UCI.

- Host
- Query Parameter
- Path Segment
- Cookie
- Protocol
- Header

If the content that your application serves is not dependent on the presence or value of these parameters in an HTTP request, you should create a rule so that when the BIG-IP assembles a UCI, it does not include those parameters.

Effective variation rules can result in a significant resource savings. For example, if your site uses a query parameter that provides session tracking information, and your site's pages are not affected by the value set for the session tracking query parameter, you can configure a variation rule to identify the session tracking query parameter as not significant for content. This way, the BIG-IP caches one version of the page for any

value of the session tracking query parameter. Without a variation rule, the BIG-IP creates one unique compiled response for every page that every user views. This results in an unnecessarily large cache.

User-specific content

By default, the BIG-IP does not include the following HTTP request elements in the UCI.

- Method
- Cookie
- User Agent
- Referrer
- Header
- Client IP

You must create a variation rule if the content you want to serve is dependent on one of these elements. That is, configure a variation rule if you want to serve different content based on.

- The method that a request uses
- The state of a cookie contained in a request
- The connecting client's IP address
- The state of the `HTTP USER_AGENT`, `REFERER`, or other request headers

If you do not create a variation rule in these situations, the BIG-IP might not provide the content you want when servicing a request.

Definition of variation rules parameters

When you define a variation rule, you specify one of the following behaviors for a parameter.

- HTTP request elements that match the variation rule result in unique page content. The BIG-IP uses the specified parameter and its value in the UCI it creates for the request.
- HTTP request elements that match the variation rule do not result in unique page content. The BIG-IP ignores the value set for the specified parameter, which means that the parameter and its value are not included in the UCI that the BIG-IP creates for the request.

Variation rules affect the UCI independently of whether the response was cached. Therefore, the BIG-IP applies variation rules to all requests after it matches the request to a node, even when it sends the request to the origin server for fresh content.

Value groups

A *value group* is a collection of values for a variation rule parameter, enabling you to define several different parameter values for the same variation rule. Each value can prompt a different behavior by the BIG-IP system.

For example, a matching rule based on `cookie A` can only specify a single set of values for the cookie, depending on a match or a no match. For more flexibility, you could use a variation rule with a value group for `cookie A`, consisting of several rules such as these examples.

- When `cookie A` begins with `aa`, the page content is the same. This rule indicates that the BIG-IP system matches all requests to the same page if the request contains a cookie called `A` with a value that begins with `aa`.

- When `cookie A` begins with `bb`, the page content is the same. This rule indicates that the BIG-IP system matches all requests to the same page if the request contains a cookie called `A` with a value that begins with `bb`. However, the page it matches is different from the page for `cookie A` with a value of `aa`.
- For all other `cookie A` values, page content is unique. This rule indicates that the BIG-IP system matches all requests that contain a cookie called `A` with any value that does not begin with `aa` or `bb`, to a unique page specified for each cookie value. That is, if there were three requests with three different values for `cookie A` (such as `ca233`, `ca234`, `ba234`), the BIG-IP system would match the requests to three different pages.

Management of conflicting rules parameters

If a request matches two or more variation rule parameters, and the parameters define conflicting behavior (one parameter indicates that the value is significant for content and the other does not), the BIG-IP considers the parameter significant for content. This means that the BIG-IP compiles a separate response for the conflicting parameter value in the request.

If a request element matches a variation rule that contains conflicting named and unnamed query parameter values, the BIG-IP considers the query parameter ambiguous because it is not clear which value the BIG-IP should use when processing the request. For example, consider a variation rule with the parameters configured as outlined in the following table.

Table 8: Example of conflicting query parameters

Parameter	Value Match	Content to Serve
All other query parameters	All values	different content
Unnamed query parameter in ordinal 1	All values	same content

When applying this variation rule to the request for

`http://www.siterequest.com/show.jsp?computer&vendor=whiteBox`, the BIG-IP could interpret `computer` as either of the following.

- A query parameter named `computer` that has no value set for it.
- An unnamed query parameter in ordinal 1, for which the value is set to `computer`.

In this example, it is unclear whether the BIG-IP should use `computer` in the UCI, and whether the result is a unique page. Therefore, the BIG-IP determines that the query is ambiguous, and matches the request to the node with the highest priority on the Policy Tree.

By default the BIG-IP treats all ambiguous query parameters as a named query parameter without a value. You can, however, override this default behavior.

Managing Compiled Responses with Assembly Rules

Overview: Assembly rules

The BIG-IP® device manages content in the form of compiled responses in cache, which contain applets that the BIG-IP device uses to assemble pages. The BIG-IP device determines how to assemble content by using the following assessments from assembly rules.

- Which content assembly features to apply, such as Intelligent Browser Referencing and MultiConnect.
- How to change parameter values for embedded URLs by substituting values from the request, or by randomly generated values.

Because the BIG-IP device performs content assembly after it matches the response to a node, it always uses the matched response node's assembly rules, which it caches as part of the original compiled response. If the response matches to a different node than the request, the BIG-IP device considers the assembly rules associated with the request's node as irrelevant.

Management of content served from origin web servers

The primary purpose of the **Enable Content Assembly on Proxies** setting is to allow the BIG-IP® device to compress content as required, and to manage content using the Intelligent Browser Referencing feature, even if the content is not served from cache. Using content compression or Intelligent Browser Referencing, without selecting the **Enable Content Assembly on Proxies** check box, applies compression to HTML documents and Intelligent Browser Referencing to links in HTML documents that are served from cache, but not to any HTML documents that bypass the cache.

***Note:** If a policy enables a document to be cached, but the document is not yet cached or has expired, the BIG-IP services from the system's cache and performs content assembly even when the **Enable Content Assembly on Proxies** option is disabled.*

Proxying Requests and Responses

Overview: Proxying rules

In general, the BIG-IP system attempts to service all HTTP requests from the system's cache. However, if you have certain types of content that you do not want the BIG-IP system to service from the system's cache, you can configure proxying rules. Using proxying rules, you identify HTTP request elements that prompt the BIG-IP system to send a request to your origin web servers for content.

You configure proxying rules from the proxying screen.

Proxying rules parameters

The proxying rule parameters consist of several components.

- **Proxying Options.** This setting provides the following options.
 - **Always proxy requests for this node.** When you select this option, the BIG-IP sends all requests that match the associated node to the origin web server for content. This option overrides any configured proxying rules.
 - **Configure and use Proxy Rules for this node.** When you select this option, the BIG-IP applies configured proxy rules to requests that match the associated node.
- **BIG-IP Cache Mode.** You can select one of the following caching modes for the node to cache objects defined by override rules.
 - **Memory & Disk Cache.** When you select this setting, the BIG-IP caches objects for the selected node to memory and disk cache.
 - **Memory-only Cache.** When you select this setting, the BIG-IP caches objects for the selected node only to memory. In the event of power loss, memory is cleared, providing greater security.
- **Proxy Rules.** For this option, you can define specific parameters for proxying rules. In general, proxy rules options are only relevant to requests that match their node, rather than to matched responses.
- **Proxy Override Rules.** For this option, you can define parameters and associated conditions under which the BIG-IP should ignore proxying rules options.

Managing Requests and Responses with Lifetime Rules

Overview: Lifetime rules

The length of time that a client browser, upstream device, or BIG-IP system keeps compiled content in its cache before refreshing it is called *content lifetime*. Content lifetime is expressed in the form of a time to live (TTL) value, and can vary for each cached response.

When content is in cache longer than its TTL value, the BIG-IP system considers the content expired. When the BIG-IP system receives a request for expired content, it sends that request to the origin web servers for fresh content, replaces the expired cached content with the fresh response, and then responds to the request.

Lifetime managed requests

The HTTP Cache-Control version 1.1 header specification identifies headers that are used to control web entities, such as the BIG-IP. Clients that are HTTP version 1.1-compliant are capable of providing request headers, which contain directives that control caching behavior.

If the **Honor Headers In Request** setting is enabled, the BIG-IP obeys any HTTP Cache-Control header directives configured for the client. If this setting is enabled and the client includes an HTTP Cache-Control header directive that indicates the client is not willing to accept content served from a cache, the BIG-IP refreshes the corresponding cached content for the request, even if that content has not yet expired.

If you want to use HTTP lifetime headers, but want to be able to serve valid content that the BIG-IP cached when applicable, you can move the **no_cache** list item for **Honor Headers In Request**, in the BIG-IP Cache Settings area, from the **Selected** list to the **Available** list.

Lifetime managed responses

BIG-IP lifetime rules only apply to responses; they are not relevant to requests. The BIG-IP applies lifetime rules only to responses that it receives from the origin web server. You can define how the BIG-IP manages cached content for responses, through the following lifetime rule settings.

Lifetime Rule Settings	Description
WebAccelerator Cache Settings	These settings specify how long the BIG-IP retains cached content, how long the BIG-IP serves cached content if the origin web server is not available, and when to expire cached content. These settings also specify whether the BIG-IP honors the TTL values provided with the headers in the request and provided with the headers sent from the origin web server.
Client Cache Settings	These settings specify whether the client browser (or other upstream device) should store content locally and, if so, the maximum time the browser should store content. They also specify whether the BIG-IP should honor any existing <code>no-cache</code> directives, use custom Cache-Control directives, or replace origin web server directives with a <code>no-cache</code> directive.

These settings apply to any HTTP response that matches to a leaf node for a specific acceleration policy, for which the option is set.

About specifying the amount of time to store cached content

If you have certain content that rarely changes, you can increase the amount of time that the BIG-IP and client store that content. This reduces the load on your origin web server and increases the perceived performance of your site.

The value under **WebAccelerator Cache Settings** for the **Maximum Age** setting determines how long the BIG-IP stores cached content. The values under **Client Cache Settings** for the **Maximum Age** and **S-Max Age** settings determine how long the client stores cached content. The **S-Max Age** setting can only be used for a shared cache. When the limit is met, the BIG-IP or client browser requests fresh content from the origin web server.

***Note:** Specified **Maximum Age** and **S-Max Age** header settings are used only when they are not provided in a response from the origin web server, or when they are not configured to be honored by the BIG-IP.*

Specifying the amount of time for the BIG-IP to store cached content

Under **WebAccelerator Cache Settings**, the **Maximum Age** can be configured either as a heuristic value or as a specific amount of time.

For **Use HTTP Lifetime Heuristic**, the **%Heuristic** value is a percentage of time that the BIG-IP uses to calculate the lifetime for cached content from the last time it was refreshed. To determine the lifetime based on this setting, the BIG-IP reviews the value for the HTTP `LAST_MODIFIED` response header, and computes the cache expiration according to the value specified for the **%Heuristic** percentage. The formula for this calculation is as follows: $((\text{current time} - \text{HTTP LAST_MODIFIED response header} = X) * (\text{lifetime heuristic})) + (\text{current time}) = \text{content expiration}$

For example, if the HTTP `LAST_MODIFIED` response header specifies that the object was last modified at 9:00 a.m., the current time is 11:00 a.m., and the **%Heuristic** setting is a value of 50, then the content expiration is 12:00 p.m.

The **Use HTTP Lifetime Heuristic** setting is only in effect if you are using HTTP headers to identify content lifetime. Use this setting only if you want to use the HTTP `LAST_MODIFIED` response header to set compiled response TTL values.

***Note:** If the origin web server does not provide an HTTP `LAST_MODIFIED` response header or value, the BIG-IP will use an internal value to calculate the lifetime for cached content.*

Clearing the **Use HTTP Lifetime Heuristic** check box enables you to configure a specific amount of time for the **Maximum Age** value, with units of time ranging from **Seconds** through **Days**. Setting the **Maximum Age** value to 0 initiates a refresh for each request, preventing the BIG-IP from caching the associated content. A **Maximum Age** setting of 0 can significantly reduce the load on the origin web server, especially for large files and frequently accessed files, because the BIG-IP initiates a refresh for each request, instead of full GET request.

Specifying the amount of time for a client to store cached content

Under **Client Cache Settings**, the **Preserve Origin Web Server headers/directives to downstream devices** option and the **Custom Cache-Control Directives** option use the **Maximum Age** and **S-Max Age** settings to configure a specific amount of time, with units of time ranging from **Seconds** through **Days**.

Setting the **Maximum Age** or **S-Max Age** value to 0 initiates a refresh for each request, preventing the client from caching the associated content.

About serving cached content when origin web server content is unavailable

The BIG-IP can continue to serve content from its cache when an origin web server responds with specific HTTP status codes. By specifying a **Stand-in Period** and a **Stand-in Code**, you can enable the BIG-IP to provide stale content for those specific conditions.

Stand-in Period

In the **WebAccelerator Cache Settings** area, the value for the **Stand-in Period** setting identifies how long the BIG-IP continues to serve content from cache if the origin web server responds to the BIG-IP's requests for fresh content with an HTTP 404 (page not found), 500 (internal server error), or 504 (service unavailable) response. If the BIG-IP cannot retrieve fresh content from the origin web server after the stand-in period expires, or if the stand-in period is undefined, the BIG-IP provides the HTTP 404, 500, or 504 response from the origin web server.

The stand-in period requires an origin web server response. If the origin web server is down, and provides no response, the stand-in period is not applied.

Stand-in Code

If network failure occurs, or if the origin web server responds with certain error codes, the BIG-IP cannot validate cache with the origin web server. However, you can configure the BIG-IP to serve stale content during these conditions, by using the **WebAccelerator Cache Settings Stand-in Code** settings. The BIG-IP serves invalid content to the client if the origin web server response code meets these conditions. If a stand-in response code is not configured, the BIG-IP uses default values (404, 500, or 504) to determine whether it is serving stale content.

About preserving origin web server headers and directives to downstream devices

When you select the **Preserve origin web server headers/directives to downstream devices** option, under **Client Cache Settings**, the BIG-IP directs the client browser to store content locally on the client, in accordance with the cache directives that are defined in the HTTP headers sent from the origin web server.

To preserve a cache control directive and send it to downstream devices, you must include the directive for **Origin Web Server Headers** in the **Selected** list.

Note: The BIG-IP inserts an *Expires* header into the response when *expires* is selected, if the origin web server does not include an *Expires* header. It does not insert an *Expires* header into the response when *all* is selected, if the origin web server does not include an *Expires* header.

You can also specify a **Maximum Age** or an **S-Max Age** value to use when these directives are not specified by the origin web server. The BIG-IP uses the specified **Maximum Age** and **S-Max Age** value only when the origin web server value is not provided.

When you add `Cache-Control` extensions to the **Custom Cache Extensions** list, the BIG-IP preserves the specified extensions to send to the client. Unless specified, any `Cache-Control` extensions from the origin web server are removed from the response.

Custom Cache-Control directives

When you select the **Custom Cache-Control directives** option in **Client Cache Settings**, the BIG-IP enables you to do the following:

- Specify which origin web server headers to preserve and send to the client.
- Apply specified **Maximum Age** and **S-Max Age** headers to the specified origin web server headers.

***Note:** The specified **Maximum Age** and **S-Max Age** headers are used only when they are not provided by the origin web server.*

- Preserve `Cache-Control` extension headers sent from the origin web server.

When you add `Cache-Control` extension headers to the **Custom Cache Extensions** list, the BIG-IP preserves the specified extensions to send to the client. Unless specified, any `Cache-Control` extensions from the origin web server are removed from the response.

About replacing origin web server headers and directives with a no-cache directive

In **Client Cache Settings**, when you select the **Replace Origin Web Server Headers/Directives with no-cache** option, the BIG-IP inserts a `no-cache` directive into the HTTP `Cache-Control` header that is returned from the origin web server. This header instructs the client browser to not cache content.

When you add `Cache-Control` extensions to the **Custom Cache Extensions** list, the BIG-IP preserves the specified extensions to send to the client. Unless specified, any `Cache-Control` extensions from the origin web server are removed from the response.

Invalidating Cached Content

Overview: Invalidating cached content for an application

You can manually invalidate cached content for one or more applications, which expires, but does not remove, the cached content. Invalidating cached content by application is useful for troubleshooting as well as for manually refreshing the cached content.

Overview: Invalidating cached content for a node

Cache invalidation is a powerful tool that you can use to maintain tight coherence between the content on your origin web servers and the content that the BIG-IP system caches.

If you update content for your site at regular intervals, such as every day or every hour, you can use lifetime rules to ensure that the system's cache is refreshed with the same frequency. Invalidation rules, however, allow you to expire cached content before it has reached its time to live (TTL) value, and is a good tool to use when content updates are event-driven, such as when an item is added to a shopping cart, a request contains a new auction bid, or a poster has submitted content on a forum thread.

When you configure invalidation rules, you define elements in a request that prompt the BIG-IP system to invalidate and refresh specific cached content. When the BIG-IP system receives a request that matches the parameters that you specified for the invalidation rule, it performs the following steps.

- Invalidates the cached content that it would have served.
- Sends the request to the origin web server for fresh content.
- Replaces the specified content, which it previously had in cache, with the new content it receives from the origin web server.
- Responds to the request with the refreshed content.

You can create invalidation rules that are based on a specific parameter, for example, an invalidation rule based on a certain cookie.

Important: *Although there might be situations that require you to invalidate a significant portion of the cache, it is important to keep in mind that such a broad invalidation process can tax the origin web server as it attempts to respond to multiple requests for new content. For this reason, F5 Networks® recommends that you make the invalidation rule parameters as specific as possible, whenever possible.*

Invalidations triggers

The BIG-IP triggers an invalidation rule for a request, only when the following conditions exist.

- An invalidation rule is active.

You can create invalidation rules and enable or disable them at any time.

- The invalidation rule contains a path parameter for both the **Request Header Matching Criteria** and the **Cached Content to Invalidate** settings.

If you do not want to assign a specific path, you can use a single slash (/).

- The invalidations rule has reached its effective time.

You can specify that invalidations rules are effective immediately, or you can set a time in the future.

- The BIG-IP has refreshed the cached content that corresponds to the request, before the effective date on the invalidations rule.

This ensures that the BIG-IP module does not invalidate a compiled response more than once, under the same invalidations rule. A compiled response's refresh time identifies the last time the BIG-IP refreshed that compiled response with content from the origin web server. If the compiled response was refreshed after the invalidations rule went into effect, the BIG-IP considers the content current.

- The request matches the configured request header matching criteria that you specified. The information that appears on the request must match all the HTTP request data type parameters that you specified for the Request Header Matching Criteria within the invalidations rule.

For example, the following requests are candidates for invalidation for an invalidations rule that specifies, in the Request Header Matching Criteria, that the product query parameter must be Computers and that the path for the request must begin with /apps.

```
http://www.somesite.com/apps/shop.jsp?action=show&product=Computers
```

```
http://web1.somesite.com/apps/search/simple.jsp?product=Computers&
category=desktop
```

While the following requests are not candidates for invalidation.

```
http://www.somesite.com/shop.jsp?action=show&product=Computers
```

```
http://web1.somesite.com/apps/search/simple.jsp?product=Organizers&
category=desktop
```

Invalidations lifetime

Invalidations rules are typically targeted at a range of compiled responses. The BIG-IP does not invalidate a compiled response until it matches a request to the invalidations rule for that compiled response, and it does not invalidate a range of compiled responses until it receives a request for each individual response in the range.

By assigning an appropriate lifetime for a rule, the BIG-IP ensures that it refreshes every targeted compiled response, before it discards the rule. The BIG-IP assigns the lifetime value for the invalidations rule by examining the maximum lifetime values set for all compiled responses targeted by the rule, and using the longest TTL value it finds. When the longest TTL value for the compiled responses is reached, the BIG-IP considers those compiled responses expired and refreshes them, even if an invalidation trigger has not occurred.

For example, if you created an invalidations rule for any requests that have either /apps or /srch in their path, your Policy Tree would include two nodes with application matching rules set so that requests with /apps in the path match to one node, and requests with /srch match to the other node. For this example, the /srch node has a lifetime rule specifying a maximum age of 15 minutes, while the /apps node uses the system's maximum lifetime of 24 hours.

Compiled responses that the BIG-IP creates to service requests matching the /srch node have a maximum lifetime of 15 minutes. Compiled responses that the BIG-IP creates to service requests matching the /apps node uses a maximum lifetime of 24 hours. Based on this, the BIG-IP assigns a 24-hour lifetime for the invalidations rule.

During the 24 hours that the rule is in effect, the BIG-IP refreshes compiled responses either because they match an invalidations rule or because they have exceeded the set lifetime value. Either way, the BIG-IP refreshes any content matching the invalidations rule at least once before it discards the rule.

Invalidations rules parameters

When configuring an invalidations rule, you must specify parameters for the following settings.

- **Request Header Matching Criteria.** You define the parameters that the BIG-IP must match in a request, in order to trigger the invalidations rule.
- **Cached Content to Invalidate.** You define the content to invalidate and refresh, if the BIG-IP finds the parameters in the HTTP request header that are specified in the **Request Header Matching Criteria** setting.

Important: When configuring an invalidation rule, all parameters are optional except for the **Path** parameter. If you do not specify the **Path** parameter for the **Request Header Matching Criteria** and the **Cached Content to Invalidate** settings, the invalidations rule does not trigger the BIG-IP to invalidate the specified cache. If you do not want to define a specific path, you can use a single slash (/).

Request header matching criteria

For the **Request Header Matching Criteria** setting, you specify the HTTP request data type parameter that the BIG-IP must find in an HTTP request header, in order to trigger the invalidations rule. For the BIG-IP to apply an invalidations rule, the HTTP request must match to a leaf node as well the associated parameters specified for the **Request Header Matching Criteria** setting.

Not all requests that match a node that you define will trigger the invalidations rule. You can define a subset of matching requests by specifying additional parameters for the source as part of the rule. Then, only the defined subset of requests trigger the invalidations rule.

For example, assume that to match to a particular node, a request must include `/apps/shopping` in the path and the query parameter `cart` must be present. To configure the **Request Header Matching Criteria** setting parameters for this node, you specify that a request must include the query parameter `cart` and the value of `cart` must equal `Add`. In this case, not all requests that match the node prompt the invalidation. The BIG-IP only invalidates requests where `cart=add`.

Cached content to invalidate

The BIG-IP only matches a request to an invalidation rule if it first finds the parameters set for the **Request Header Matching Criteria** setting. If it matches a request to the parameters configured for the **Request Header Matching Criteria** setting, only then does the BIG-IP review the parameters set for the **Cached Content to Invalidate** setting. If a match occurs, the BIG-IP invalidates the content specified, and retrieves fresh content from the origin web server. The BIG-IP stores the fresh content in cache and services the request. When configuring an invalidations rule, you must provide at least one parameter based on the **Path** data type for the **Cached Content to Invalidate** setting.

Managing Object Types

Overview: Object classification

Before sending a response to a client, the BIG-IP system enters an informational `X-WA-Info` response header into the response to describe how it handled the response. You cannot change these informational headers, and they do not affect processing, however, they can provide useful information for evaluating the efficiency of your acceleration policies.

Part of the information included in the `X-WA-Info` response header is the object type. The BIG-IP system classifies, by object type and group, every response it receives from the origin web servers. The object type and group classification determine how the BIG-IP system handles compression for the response.

Classification by object type

To classify a response by object type, the BIG-IP reviews the response headers and classifies the responses based on the first information it finds. The following list defines the order for classification.

- File extension in the Content-Disposition header's file name field
- File extension in the Content-Disposition header's extension field
- Content-Type header in the response, unless it is an ambiguous MIME type
- Extension of the path in the request

For example, if the extension in the Content-Disposition header's file name field is empty, then the BIG-IP looks at the Content-Disposition header's extension field. If Content-Disposition header's field has an extension, the BIG-IP checks to see if an object type is configured for the extension. If there is no match, it assigns an object type of other, and uses the object settings for other. The BIG-IP looks at the information in the Content-Type header only if there is no extension in the Content-Disposition header's file name or extension fields.

Classification by group

In addition to classifying the response by object type, the BIG-IP also classifies the response by group.

For example, in the following `X-WA-Info` response header the object type (OT) is defined as Microsoft Word (mword) and the object group (OG) is documents.

```
X-WA-Info: [S10101.C30649.A28438.RA0.G0.U58517886].[OT/mword.OG/documents]
```

Management of object types

The BIG-IP offers two object types.

- **Pre-defined Object Types.** The BIG-IP ships with several predefined object types, most of which are optimized for objects associated with specific applications.

- **User-defined Object Types.** A user-defined object type is an object type that you create and for which you specify all of the parameters dictating how the BIG-IP manages the specified object type.

The **Objects Types** screen displays all of the object types that the BIG-IP system is currently applying to your acceleration policies. From the **Object Types** screen, you can view the object types that the BIG-IP is currently applying to acceleration policies, as well as access additional screens where you can perform the following tasks.

- Create a user-defined object type.
- View and modify the settings for an existing user-defined or predefined object type.
- Delete a user-defined object type.

Note: *You can delete only user-defined object types; you cannot delete predefined object types.*

When you create a new object type or modify an existing object type, the BIG-IP system applies the object type changes globally to all acceleration policies.

When you modify a predefined object type and save it, an information icon displays next to the display name in the predefined object types table, indicating that the parameters for the object type are modified from the original version that was shipped with the BIG-IP device.

Caching Objects in a VIPRION Cluster

Overview: Acceleration in a cluster

A VIPRION® system provides you with the ability to cache objects either for a policy node in a cluster or on a single cluster member. Typically, caching objects in a cluster achieves optimum acceleration for large, static objects. Comparatively, caching objects on a single cluster member achieves optimum acceleration for small, dynamic objects.

Immediately Caching Dynamic Objects

Overview: Caching an object on first hit

The BIG-IP[®] system provides you with the ability to cache an object on the first cache hit, that is, the first time that an object is seen by the BIG-IP system. Typically, the BIG-IP system waits until the object is known to be popular, allowing a limited number of hits against the original content. Caching the object on first hit, however, causes the BIG-IP system to immediately cache the object, even if it is not popular. You only want to apply this setting for highly dynamic objects (for example, stock quotes provided by a stock ticker) that you want to cache for a limited time to offload the origin web server. If you use this setting on content that omits a `Content-Length` header, or if the object is an HTML, JavaScript (JS), or cascading style sheet (CSS) object, a significant degradation in performance can occur.

Accelerating Parallel HTTP Requests

Overview: HTTP request queuing

You can use the BIG-IP® system to accelerate responses and reduce the number of requests sent to origin web servers, freeing them to perform other tasks, by queuing HTTP requests. *HTTP request queuing* provides the ability to queue a large number of parallel HTTP requests for the same object, and provide a cached response to those requests, resulting in accelerated responses from the BIG-IP system and a reduction in requests sent to origin web servers.

The BIG-IP system manages the queued HTTP requests in accordance with the cache status of the requested object, that is, whether the requested object is uncached, cached and valid, cached and expired, uncacheable, or nonexistent.

Object cache status	Description
Initial requests for an uncached object	When the BIG-IP system receives a large number of parallel requests for an object that is not yet cached, it queues the requests, and then sends one of the requests to an origin web server. When the BIG-IP system receives the first response, it determines whether the response is cacheable while sending the response to the client. If the response is cacheable, the BIG-IP system sends a second request to the origin web server, caches the response with the object, and uses that cached response to service the queued requests.
Requests for a cached object	When the BIG-IP system receives a large number of parallel requests for a valid cached object, it services the requests with the cached response.
Requests for an expired cached object	If a cached object is expired, instead of sending all requests to the origin web server, the BIG-IP system queues the requests, and then sends one request to an origin web server for fresh content. When the BIG-IP system receives the fresh response, it caches the response with the fresh content, and uses the cached response to service the queued requests.
Requests for an invalidated cached object	If a cached object requires validation, the BIG-IP system can queue the requests, and then send one request to an origin web server for fresh content. When the response is received, the BIG-IP system caches the response with the fresh content, and uses the cached response to service the queued requests.
Requests for an uncacheable object	Sometimes, an object cannot be cached, for example, if the object exceeds the maximum object size or if the response includes a <code>no-store</code> response header. When the BIG-IP system first receives a large number of parallel requests for an object that cannot be cached, instead of sending each request to an origin web server, the BIG-IP system queues the requests, and then sends one request to an origin web server. When the BIG-IP system receives the response, it sends the queued requests to the origin web server. Subsequent requests for the uncacheable object bypass the BIG-IP system and are sent directly to the origin web server.

Object cache status	Description
Requests for a nonexistent object	When the BIG-IP system receives a large number of parallel requests for an object that does not exist or no longer exists, the BIG-IP system can queue the requests, and then send one request to an origin web server. When the BIG-IP system receives the response with a 404 (Not Found) status code, it services the queued requests with the 404 (Not Found) response. Note that the 404 (Not Found) response is not cached, and all subsequent requests for the nonexistent object are sent to the origin web server.

Managing HTTP Traffic with the HTTP/2 Profile

Overview: Managing HTTP Traffic with the HTTP/2 profile

You can configure the BIG-IP® Acceleration HTTP/2 profile to provide gateway functionality for HTTP 2.0 traffic, minimizing the latency of requests by multiplexing streams and compressing headers.

A client initiates an HTTP/2 request to the BIG-IP system, the HTTP/2 virtual server receives the request on port 443, and sends the request to the appropriate server. When the server provides a response, the BIG-IP system compresses and caches it, and sends the response to the client.

Important: *The BIG-IP system supports HTTP/2 for client-side connections only. This means that when a client that supports HTTP/2 connects to a virtual server that has an HTTP/2 profile assigned to it, the resulting server-side traffic (such as traffic sent to pool members) is sent over HTTP/1.1.*

The HTTP/2 profile does not support source address persistence.

Summary of HTTP/2 profile functionality

By using the HTTP/2 profile, the BIG-IP system provides the following functionality for HTTP/2 requests.

Creating concurrent streams for each connection.

You can specify the maximum number of concurrent HTTP requests that are accepted on a HTTP/2 connection. If this maximum number is exceeded, the system closes the connection.

Limiting the duration of idle connections.

You can specify the maximum duration for an idle HTTP/2 connection. If this maximum duration is exceeded, the system closes the connection.

Enabling a virtual server to process HTTP/2 requests.

You can configure the HTTP/2 profile on the virtual server to receive HTTP, SPDY, and HTTP/2 traffic, or to receive only HTTP/2 traffic, based in the activation mode you select. (Note the HTTP/2 profile to receive only HTTP/2 traffic is primarily intended for troubleshooting.)

Inserting a header into the request.

You can insert a header with a specific name into the request. The default name for the header is X-HTTP/2.

Important: *The HTTP/2 protocol is incompatible with NTLM protocols. Do not use the HTTP/2 protocol with NTLM protocols.*

About HTTP/2 profiles

The BIG-IP® system's Acceleration functionality includes an HTTP/2 profile type that you can use to manage HTTP/2 traffic, improving the efficiency of network resources while reducing the perceived latency of requests and responses. The Acceleration HTTP/2 profile enables you to achieve these advantages by multiplexing streams and compressing headers with Transport Layer Security (TLS) or Secure Sockets Layer (SSL) security.

The HTTP/2 protocol uses a binary framing layer that defines a frame type and purpose in managing requests and responses. The binary framing layer determines how HTTP messages are encapsulated and transferred between the client and server, a significant benefit of HTTP 2.0 when compared to earlier versions.

All HTTP/2 communication occurs by means of a connection with bidirectional streams. Each stream includes messages, consisting of one or more frames, that can be interleaved and reassembled using the embedded stream identifier within each frame's header. The HTTP/2 profile enables you to specify a maximum frame size and write size, which controls the total size of combined data frames, to improve network utilization.

Multiplexing streams

You can use the HTTP/2 profile to multiplex streams (interleaving and reassembling the streams), by specifying a maximum number of concurrent streams permitted for a single connection. Also, because multiplexing streams on a single TCP connection compete for shared bandwidth, you can use the profile's Priority Handling settings to configure stream prioritization and define the relative order of delivery. For example, a Strict setting processes higher priority streams to completion before processing lower priority streams; whereas, a Fair setting allows higher priority streams to use more bandwidth than lower priority streams, without completely blocking the lower priority streams.

Additionally, you can specify the way that the HTTP/2 profile controls the flow of streams. The Receive Window setting allows HTTP/2 to stall individual upload streams, as needed. For example, if the BIG-IP system is unable to process a slow stream on a connection, but is able to process other streams on the connection, it can use the Receive Window setting to specify a frame size for the slow stream, thus delaying that upload stream until the size is met and the receiver is able to process it, while concurrently proceeding to process frames for another stream.

Compressing headers

When you configure the HTTP/2 profile's Header Table Size setting, you can compress HTTP headers to conserve bandwidth. Compressing HTTP headers reduces the object size, which reduces required bandwidth. For example, you can specify a larger table value for better compression, but at the expense of using more memory.

HTTP/2 profile settings

This table provides descriptions of the HTTP/2 profile settings.

Setting	Default	Description
Name		Specifies the name of the HTTP/2 profile.
Parent Profile	http2	Specifies the profile that you want to use as the parent profile. Your new profile inherits all settings and values from the parent profile specified.
Concurrent Streams Per Connection	10	Specifies the number of concurrent requests allowed to be outstanding on a single HTTP/2 connection.
Connection Idle Timeout	300	Specifies the number of seconds an HTTP/2 connection is left open idly before it is closed.
Insert Header	Disabled	Specifies whether an HTTP header that indicates the use of HTTP/2 is inserted into the request sent to the origin web server.
Insert Header Name	X-HTTP/2	Specifies the name of the HTTP header controlled by the Insert Header Name setting.

Setting	Default	Description
Activation Modes	Select Modes	Specifies how a connection is established as a HTTP/2 connection.
Selected Modes	ALPN NPN	Used only with an Activation Modes selection of Select Modes , specifies the extension, ALPN for HTTP/2 or NPN for SPDY, used in the HTTP/2 profile. The order of the extensions in the Selected Modes Enabled list ranges from most preferred (first) to least preferred (last). Clients typically use the first supported extension. At least one HTTP/2 mode must be included in the Enabled list. The values ALPN and NPN specify that the TLS Application Layer Protocol Negotiation (ALPN) and Next Protocol Negotiation (NPN) will be used to determine whether HTTP/2 or SPDY should be activated. Clients that use TLS, but only support HTTP will work as if HTTP/2 is not present. The value Always specifies that all connections function as HTTP/2 connections. Selecting Always in the Activation Mode list is primarily intended for troubleshooting.
Priority Handling	Strict	Specifies how the HTTP/2 profile handles priorities of concurrent streams within the same connection. Selecting Strict processes higher priority streams to completion before processing lower priority streams. Selecting Fair enables higher priority streams to use more bandwidth than lower priority streams, without completely blocking the lower priority streams.
Receive Window	32	Specifies the <i>receive window</i> , which is HTTP/2 protocol functionality that controls flow, in KB. The receive window allows the HTTP/2 protocol to stall individual upload streams when needed.
Frame Size	2048	Specifies the size of the data frames, in bytes, that the HTTP/2 protocol sends to the client. Larger frame sizes improve network utilization, but can affect concurrency.
Write Size	16384	Specifies the total size of combined data frames, in bytes, that the HTTP/2 protocol sends in a single write function. This setting controls the size of the TLS records when the HTTP/2 protocol is used over Secure Sockets Layer (SSL). A large write size causes the HTTP/2 protocol to buffer more data and improves network utilization.
Header Table Size	4096	Specifies the size of the header table, in KB. The HTTP/2 protocol compresses HTTP headers to save bandwidth. A larger table size allows better compression, but requires more memory.

Managing HTTP Traffic with the SPDY Profile

Overview: Managing HTTP traffic with the SPDY profile

You can use the BIG-IP® Acceleration SPDY (pronounced "speedy") profile to minimize latency of HTTP requests by multiplexing streams and compressing headers. When you assign a SPDY profile to an HTTP virtual server, the HTTP virtual server informs clients that a SPDY virtual server is available to respond to SPDY requests.

When a client sends an HTTP request, the HTTP virtual server, with an assigned iRule, manages the request as a standard HTTP request. It receives the request on port 80, and sends the request to the appropriate server. When the BIG-IP provides the request to the origin web server, the virtual server's assigned iRule inserts an HTTP header into the request (to inform the client that a SPDY virtual server is available to handle SPDY requests), compresses and caches it, and sends the response to the client.

A client that is enabled to use the SPDY protocol sends a SPDY request to the BIG-IP system, the SPDY virtual server receives the request on port 443, converts the SPDY request into an HTTP request, and sends the request to the appropriate server. When the server provides a response, the BIG-IP system converts the HTTP response into a SPDY response, compresses and caches it, and sends the response to the client.

Note: Source address persistence is not supported by the SPDY profile.

Summary of SPDY profile functionality

By using the SPDY profile, the BIG-IP system provides the following functionality for SPDY requests.

Creating concurrent streams for each connection.

You can specify the maximum number of concurrent HTTP requests that are accepted on a SPDY connection. If this maximum number is exceeded, the system closes the connection.

Limiting the duration of idle connections.

You can specify the maximum duration for an idle SPDY connection. If this maximum duration is exceeded, the system closes the connection.

Enabling a virtual server to process SPDY requests.

You can configure the SPDY profile on the virtual server to receive both HTTP and SPDY traffic, or to receive only SPDY traffic, based in the activation mode you select. (Note that setting this to receive only SPDY traffic is primarily intended for troubleshooting.)

Inserting a header into the request.

You can insert a header with a specific name into the request. The default name for the header is X-SPDY.

Important: The SPDY protocol is incompatible with NTLM protocols. Do not use the SPDY protocol with NTLM protocols. For additional details regarding this limitation, please refer to the SPDY specification: <http://dev.chromium.org/spdy/spdy-authentication>.

Task summary

SPDY profile settings

This table provides descriptions of the SPDY profile settings.

Setting	Default	Description
Name		Type the name of the SPDY profile.
Parent Profile	spdy	Specifies the profile that you want to use as the parent profile. Your new profile inherits all settings and values from the parent profile specified.
Concurrent Streams Per Connection	10	Specifies how many concurrent requests are allowed to be outstanding on a single SPDY connection.
Connection Idle Timeout	300	Specifies how many seconds a SPDY connection is left open idly before it is closed.
Activation Mode	NPN	Specifies how a connection is established as a SPDY connection. The value NPN specifies that the Transport Layer Security (TLS) Next Protocol Negotiation (NPN) extension determines whether the SPDY protocol is used. Clients that use TLS, but only support HTTP will work as if SPDY is not present. The value Always specifies that all connections must be SPDY connections, and that clients only supporting HTTP are not able to send requests. Selecting Always in the Activation Mode list is primarily intended for troubleshooting.
Insert Header	Disabled	Specifies whether an HTTP header that indicates the use of SPDY is inserted into the request sent to the origin web server.
Insert Header Name	X-SPDY	Specifies the name of the HTTP header controlled by the Insert Header setting.
Protocol Versions	All Versions Enabled	Used only with an Activation Mode selection of NPN , specifies the protocol and protocol version (http1.1 , spdy2 , spdy3 , spdy3.1 , or All Version Enabled) used in the SPDY profile. The order of the protocols in the Selected Versions Enabled list ranges from most preferred (first) to least preferred (last). Adding http1.1 to the Enabled list allows HTTP1.1 traffic to pass. If http1.1 is not added to the Enabled list, clients that do not support http1.1 are blocked. Clients typically use the first supported protocol. At least one SPDY version must be included in the Enabled list.
Priority Handling	Strict	Specifies how the SPDY profile handles priorities of concurrent streams within the same connection. Selecting Strict processes higher priority streams to completion before processing lower priority streams. Selecting Fair enables higher priority streams to use more bandwidth than lower priority streams, without completely blocking the lower priority streams.
Receive Window	32	Specifies the <i>receive window</i> , which is SPDY protocol functionality that controls flow, in KB. The receive window allows the SPDY protocol to stall individual upload streams when needed. This functionality is only available in SPDY3.

Setting	Default	Description
Frame Size	2048	Specifies the size of the data frames, in bytes, that the SPDY protocol sends to the client. Larger frame sizes improve network utilization, but can affect concurrency.
Write Size	16384	Specifies the total size of combined data frames, in bytes, that the SPDY protocol sends in a single write function. This setting controls the size of the TLS records when the SPDY protocol is used over Secure Sockets Layer (SSL). A large write size causes the SPDY protocol to buffer more data and improves network utilization.

Accelerating Requests and Responses with Intelligent Browser Referencing

Overview: Reducing conditional GET requests with Intelligent Browser Referencing

You can increase the efficiency of the client's web browser's local cache and improve perceived access to your site by enabling the *Intelligent Browser Referencing* (IBR) feature, which reduces or eliminates requests to your site for relatively static content, such as images and cascading style sheet (CSS) files.

About conditional GET requests

When an origin web server sends a response, the client's browser stores the response in its local cache. If the cached object expires, the browser makes subsequent requests for that content using a conditional GET request in the form of an extra request header field, such as *If-Modified-Since*. If the requested object is different from the content that the browser has cached, the origin web server sends a fresh copy of the object to the browser. Otherwise, the browser uses the object that is cached locally.

Although it is faster than serving the entire object each time the browser requests it, conditional GET requests can add up to a significant amount of traffic for your site. For example, if your site has several images for each page, clients might perceive a slow response time because of the large number of conditional GET requests for the image objects.

About Intelligent Browser Referencing for HTML

You can use the **Enable Intelligent Browser Referencing To** setting to manage the web browser's cache in two ways.

- By serving qualifying content with the expiration time set long enough that it is unlikely that the browser re-requests the content in the near future.
- By using an IBR tag (such as *wa*) to append a unique value into qualifying links or URLs for web pages that match the node. This value is a hash of the object and, as such, uniquely identifies the corresponding content stored in the system's cache.

The **Enable Intelligent Browser Referencing Within** setting uses an IBR tag (such as *wa*) to append a unique value to qualifying links and URLs within web pages that match the node.

For an HTML page, the BIG-IP[®] device applies IBR to the following elements and statements.

- Image tags: ``
- Script tags: `<script src="...">`
- Link tags: `<link href="...">`
- Forms whose input type is an image: `<form><input type="image" src="..."></form>`

About Intelligent Browser Referencing for cascading style sheet files

You can use the Enable Intelligent Browser Referencing To setting to prompt the BIG-IP® device to rewrite links to cascading style sheet (CSS) files on a node.

- It can serve qualifying content with the expiration time set long enough that it is unlikely that the browser re-requests the content in the near future.

***Note:** If you also select the **Enable Intelligent Browser Referencing Within** check box, the adaptive IBR lifetime for the BIG-IP application supersedes the default IBR lifetime.*

- It can use an IBR tag (such as `wa`) to append a unique value to qualifying links or URLs for style sheets that match the node. This value is a hash of the object and, as such, is guaranteed to uniquely identify the corresponding content stored in the system's cache.

When enabled, the Enable Intelligent Browser Referencing Within setting manages responses for URLs to images and for URLs to CSS files that are externally linked or imported by a CSS file in two ways.

- By using the adaptive IBR lifetime specified in the BIG-IP application, which supersedes and is shorter than a standard IBR lifetime, enabling assembly of linked image files before all of the image files are cached, and enabling the embedded image files to refresh before a client uses stale image files from the browser's cache.
- By using an IBR tag (such as `wa`) to append a unique hash value to qualifying links or URLs to images and CSS files externally linked within CSS files.

***Note:** The **Enable Intelligent Browser Referencing Within** check box applies only to CSS files that are externally linked from an HTML or CSS file, and does not apply to embedded or inline CSS elements within an HTML file. An HTML file can link to an external CSS file by means of the `LINK` element or an `@import` statement in the `STYLE` element. A CSS file can link to an external CSS file by means of an `@import` statement in the `STYLE` element.*

Within externally linked CSS files, the BIG-IP device applies IBR to the following elements and statements.

- Link tags: `<link rel=stylesheet href="style.css" type="text/css">`
- Import statements:

```
@import url("style.css");
@import url(style.css);
@import "style.css";
@import style.css;
```

About the adaptive Intelligent Browser Referencing lifetime

When you enable Intelligent Browser Referencing for a policy node, an application can apply IBR Adaptive Lifetime (typically shorter than a standard IBR lifetime) to the CSS file. This enables the assembly of linked image files before all of the image files are cached, and allows the embedded image files to refresh before a client uses stale image files from a browser's cache. You should consider the shortest lifetime needed for an image file when configuring adaptive IBR lifetime settings. You can adjust the **IBR Adaptive Lifetime** setting for an application on the Applications screen.

Intelligent Browser Referencing example

For this Intelligent Browser Referencing (IBR) example, you have three top-level nodes on the Policy Tree as follows:

- **Home.** This branch node specifies the rules related to the home page.
- **Applications.** This branch node specifies the rules related to the applications for the site, with the following leaf nodes:
 - **Default.** This leaf node specifies the rules related to non-search related applications.
 - **Search.** This leaf node specifies the rules related to your site's search application.
- **Images.** This branch node specifies the rules related to graphics images.

For this example, your site serves a simple page that consists of two image files that appear like this:

```
<html>
<head><title>example page</title></head>
<body>
  <p>The images that your site serves:</p>
  <p></p>
  <p></p>
</body>
</html>
```

When the IBR tag (in this example, `wa`) is enabled, the BIG-IP® device modifies the page like this:

```
<html>
<head><title>example page</title></head>
<body>
  <p>The images that your site serves:</p>
  <p></p>
  <p></p>
</body>
</html>
```

The IBR tag that the BIG-IP device appends to each image source URL is a hash of the image that is stored in cache. In addition, the browser receives a long expiration time for each of the image files.

As a result, the client browser conducts subsequent requests for the page with multiple actions:

- Performing a conditional `GET` request for the base page.
- Obtaining the embedded images directly from cache if the IBR tag matches.
- Requesting new images from the BIG-IP device.

If an image on the page is modified, the BIG-IP device changes the IBR tag for the image and informs the client of the change. When the client performs a subsequent conditional `GET` request for the base page and receives the refreshed page, it compares the image, and notes the difference between `image1.jpeg;wa4RR87M90` and `image1.jpeg;waRG2076ND`. This difference prompts the client to re-request the image from the BIG-IP device.

Advanced IBR settings for general options

For the **General Options** list, this table describes the **Advanced** settings and strings for the IBR Options area.

Advanced control	Default	Description
IBR Prefix	<code>;wa</code>	<p>Specifies a string that the BIG-IP® device appends to a unique value for qualifying links or URLs embedded in your web pages.</p> <hr/> <p><i>Note: If you change the IBR prefix, test thoroughly to ensure that your application functions properly.</i></p> <hr/>
IBR Default Lifetime	26 Weeks	<p>Specifies the lifetime for an Intelligent Browser Referencing (IBR) link or URL. Units of time range from Seconds through Months.</p>
IBR Adaptive Lifetime	10 Days	<p>Specifies the lifetime for an adaptive IBR link or URL within an externally linked CSS file. Units of time range from Seconds through Months.</p> <hr/> <p><i>Note: Verify that the Enable Intelligent Browser Referencing Within check box is selected before you apply an adaptive IBR lifetime.</i></p> <hr/>

Accelerating JavaScript and Cascading Style Sheet Files

Overview: Accelerating JavaScript, HTML, cascading style sheet, and inline image files

You can improve acceleration by reducing the number and sizes of cascading style sheet (CSS), HTML, and JavaScript files transferred across a network, and by improving the ability for browsers to render content. The BIG-IP[®] system uses inlining and concatenation of CSS, HTML, and JavaScript files to reduce the number and sizes of files transferred across a network, thus improving the acceleration of traffic, and uses minification and reordering to improve the speed that browsers render content.

Task summary

About minification of JavaScript, HTML, and cascading style sheet content

Minification removes extraneous white spaces, comments, and unnecessary special characters from JavaScript and cascading style sheet (CSS) files, which reduces the file sizes. The BIG-IP[®] system supports two types of minification: .

- minifying linked JavaScript, HTML, and CSS files
- minifying embedded JavaScript, HTML, and CSS content within HTML pages

The BIG-IP system caches only the minified documents.

Note: *Minification of a continuous JavaScript comment or section of white spaces, either embedded in an HTML file or in a stand-alone JavaScript file, applies only to a continuous comment or section of white spaces that is less than 1024 bytes. If this content exceeds 1024 bytes, then that content is not minified.*

About reordering cascading style sheet and JavaScript URLs and content

Reordering cascading style sheet (CSS) and JavaScript links within an HTML document can accelerate the perceived time in which a browser renders a web page. Although the actual time required to download the page remains approximately the same, the perceived time to display the page is faster.

When a CSS link appears at the top of the page, preceding the `</head>` element, a browser can progressively render the page to quickly display the content, especially beneficial for users who access content-rich pages by means of slower Internet connections.

When a JavaScript link appears at the end of the page, preceding the `</body>` element, a browser can download multiple components in parallel for each hostname and accelerate the perceived page rendering. You can specify each JavaScript link that you prefer to relocate to the end of the page, and, consequently, accelerate the perceived page rendering. Exceptions to reordering JavaScript information include JavaScript URLs and scripts that use `document.write` to insert content for the page.

About inlining documents and image data

Inlining replaces specified URLs to JavaScript and cascading style sheet (CSS) files with an inline copy of the document, and replaces specified URLs to external images with image data.

Within an HTML document, a specified link to an external JavaScript or CSS file is replaced with the style sheet content.

If a client requests an HTML document for which the response header contains a `Cache-Control: private`, `Cache-Control: no-store`, or `Vary: User-Agent` header, the BIG-IP[®] system removes the inline content from the response, and caches the inline content.

Note: *In order for content to be inlined, the inlined content must expire later than the parent content.*

Inlining Conventions

When you use inlining functionality, the following conventions provide best results.

- Inlining objects typically include stable objects that change infrequently, objects that remain unchanged for several weeks.
- The file size for an inlining object is typically small, less than 2 KB.
- When an inlined object changes on the origin web server, the respective URL resource entry must be updated on the URL Resources page.
- For a user-defined acceleration policy that includes inlining functionality, you will want to use the default HTTP lifetime settings.
- On the Lifetime screen, configure the settings as follows:
 - The duration of WebAccelerator Cache Settings must be less than Client Cache Settings.
 - The expiration times for the inlined-content Client Cache Settings must occur concurrently with or after the parent Client Cache Settings.

About concatenation of JavaScript and cascading style sheet files

Concatenation combines a specified list of JavaScript (JS) or Cascading Style Sheet (CSS) files into a single concatenated file, which reduces the number of requests and responses, and the time required to transfer serialized files.

For each user-defined policy, you can specify lists of JS URLs and lists of CSS URLs for concatenation. Listings in each URL list appear in the specified order.

During the process of concatenation, the first JS and CSS URL within the HTML file that is specified in a JS or CSS URL list is replaced with the optimized URL, and each subsequent specified JS and CSS URL is removed. The TTL of a concatenated response is determined by the earliest expiration of the concatenated objects.

About DNS prefetching

DNS prefetching for HTTP

DNS prefetching improves page load time on HTML5 compliant browsers by resolving domain names to an IP address prior to a browser requesting content from third parties. When DNS pre-fetching headers are inserted by the BIG-IP[®] system, HTML5-compliant browsers can do DNS resolution of dynamic links in

the background while other items are being downloaded. This feature allows users to configure lists of DNS prefetch domains by inserting the following link tag in the head of an HTML document:

```
<link rel="dns-prefetch" href="http://www.siterequest.com/">
```

DNS prefetching for HTTPS

By default, DNS prefetching is always turned off for pages served in HTTPS to avoid leaking information about which particular document is served. Turning on Force Injection on HTTPS enables DNS prefetching specifically for the domains listed in a domain list. Turning on HTTPS Automatic Page Prefetch turns on DNS prefetching for the entire document served. Force Injection on HTTPS must be enabled in order to enable HTTPS Automatic Page Prefetch. Administrators are able to configure turning on and off DNS prefetching when serving over an HTTPS connection by inserting the following head tag:

```
<meta http-equiv="x-dns-prefetch-control" content="on">
```

Note: Turning on prefetch header insertion does not interfere with JavaScript or CSS transformations.

Establishing Additional TCP Connections with MultiConnect

Overview: Accelerating requests and responses with MultiConnect

Most web browsers create a limited number of persistent TCP connections when requesting data, which restricts the amount of content a client can receive at one time. You can provide faster data downloads to your clients using the BIG-IP[®] device's MultiConnect feature.

The *MultiConnect* feature enables you to specify unique subdomains that prompt the browser to open more persistent TCP connections (up to five per HTTP subdomain and five per HTTPS subdomain generated by the BIG-IP device). The origin web servers never get a request from these additional subdomains; they are used exclusively on externally linked URLs or links that request images or scripts and are only for requests or responses between the client and the BIG-IP device. If the BIG-IP device needs to send a request to the origin server, it removes the subdomain prefixes before sending the request.

The BIG-IP device uses the MultiConnect feature only on the following types of links:

- Image tags: ``
- Script tags: `<script src="...">`
- Forms whose input type is an image: `<form><input type="image" src="..."></form>`

Optimization of TCP connections

The BIG-IP[®] application acceleration provides MultiConnect functionality that decreases the number of server-side TCP connections required while increasing the number of simultaneous client-side TCP connections available to a browser for downloading a web page.

Decreasing the number of server-side TCP connections can improve application performance and reduce the number of servers required to host an application. Creating and closing a TCP connection requires significant overhead, so as the number of open server connections increases, maintaining those connections while simultaneously opening new connections can severely degrade server performance and user response time.

Despite the ability for multiple transactions to occur within a single TCP connection, a connection is typically between one client and one server. A connection normally closes either when a server reaches a defined transaction limit or when a client has transferred all of the files that are needed from that server. The BIG-IP system, however, operates as a proxy and can pool TCP server-side connections by combining many separate transactions, potentially from multiple users, through fewer TCP connections. The BIG-IP system opens new server-side connections only when necessary, thus reusing existing connections for requests from other users whenever possible.

The **Enable MultiConnect To** check box on the **Assembly** screen of BIG-IP applies MultiConnect functionality to image or script objects that match the node. The **Enable MultiConnect Within** check box, however, applies MultiConnect functionality to image or script objects that are linked within HTML or CSS files for the node.

MultiConnect example

For this example, your site serves a simple page (<http://www.siterequest.com/index.htm>) that consists of several image files. The page that your site serves appears as follows:

```
<html>
  <head><title>example page</title></head>
  <body>
    <p>The images that your site serves:</p>
    <p></p>
    <p></p>
    <p></p>
    <p></p>
  </body>
</html>
```

An additional subdomain prefix of `wa` is configured for the host map and the MultiConnect feature enabled, so the BIG-IP® device modifies the page and serves it as follows:

```
<html>
  <head><title>example page</title></head>
  <body>
    <p>The images that your site serves:</p>
    <p></p>

    <p></p>

    <p></p>

    <p></p>

  </body>
</html>
```


Serving Specific Hyperlinked Content with Parameter Value Substitution

Overview: Serving specific hyperlinked content with parameter value substitution

You can use parameter value substitution functionality to change a targeted parameter's value on a specific page serviced from cache, so that the client-specified parameter appears on the URL embedded in the page. This substitution is especially beneficial when a query parameter contains identification information for a site's visitors.

Some requested pages include hyperlinks that vary according to the request to provide dynamic information. For example, you can configure parameter value substitution so that a request with a query parameter called `shopper` produces HTML output with its embedded hyperlinks varying the value for `shopper`. Thus, when a query parameter contains identification information for a site's visitors, it prompts the BIG-IP® device to serve different content for the request, based on the specific visitor.

Conversely, if parameter value substitution is not configured, the BIG-IP device uses the value that it cached for the original request, for all subsequent requests after the first, even if the subsequent requests have different values that the origin web server used in the response.

About configuring value substitution parameters for an assembly rule

When you configure parameter value substitution, you specify a source definition and a target definition. You also have the option to provide a URL prefix for the target, to limit the URLs to which the BIG-IP® device performs the substitution.

Source definition

When you define a *source definition*, you specify the source of the value that you want the BIG-IP device to embed in the URL, in place of the cached (target) value. Typically, the source is a certain request element, such as a query parameter. However, you can define another source type, such as a random number, by using *number randomizer*. When you define the source, you identify the parameter by data type and name or location in the request.

If you use the request URL as the source, the BIG-IP device uses the entire absolute or relative request URL as the value to substitute.

Target definition

A *target definition* specifies the element in a URL that is replaced with the value from the source definition. The target is a specific element in the URL, such as a particular query parameter, the value for which the BIG-IP device replaces during substitution. When you define the target, you identify the parameter by data type and name or location in the URL, for example, by using a query parameter, an unnamed query parameter, or a path segment.

Although you can specify the same request element for both the source and the target, the parameter specified for the source is located in the request URL and the parameter specified for the target is located in the embedded URL in the cached page.

By default, the BIG-IP device performs substitution on all embedded URLs in which the identified target appears. You have the option to limit which URLs embedded in a page are targeted for substitution by specifying a prefix that an embedded URL must match before the BIG-IP device performs substitution.

Example: Substitution of request URL with target URL

The BIG-IP device substitutes the request URL with the target URL, including it as the `url` query parameter in the embedded URL of the cached page.

In this example, the following request URL is used as a source definition:

```
http://www.siterequest.com/apps/something.jsp?entity=orange.
```

The request URL is then substituted with the following target URL: `<a`

```
href="http://www.siterequest.com/anotherthing.jsp?
```

```
url=http://www.siterequest.com/apps/something.jsp? entity=orange&....">
```

About using number randomizer for parameter value substitution

When configured, the assembly rule's number randomizer setting generates a random number and places it in a targeted location in the embedded URL. When the BIG-IP® device compiles the response, it examines the target location to see the length of the string used for the value. On subsequent page requests, the BIG-IP replaces that value with a random number of the same length.

This setting is generally used for sites that use an Internet advertisement agency, which requires random numbers to be placed on URLs that request ads as a way to interrupt traditional caches. By requiring a random number, Internet advertisement agencies force this traffic to their site.

For example, consider a cached page that includes the following embedded URL:

- `<a href src="http://www.siterequest.com/getAd?entity=45&..."> `

You can configure random value substitution so that the BIG-IP sets random values set for the entity query parameter for subsequent pages as follows:

- `<a href src="http://www.siterequest.com/getAd?entity=45&..."> `
- `<a href src="http://www.siterequest.com/getAd?entity=11&..."> `
- `<a href src="http://www.siterequest.com/getAd?entity=87&..."> `

A parameter value substitution example

The following examples show requests with and without parameter value substitution.

An example without parameter value substitution

This example describes a standard sequence without parameter value substitution to serve a cached page.

An original request generates a cached page.

Original page request

```
http://www.siterequest.com/apps/shopping.jsp? shopper=AAAnb35vnM09&.... URL in  
cached page <a href="http://www.siterequest.com/apps/shopping.jsp?  
shopper=AAAnb35vnM09&...."link</a>
```

For subsequent requests of the example page, the BIG-IP® system still serves the cached page.

Subsequent request

`http://www.siterequest.com/apps/shopping.jsp? shopper=SNkj90qcL47&....`

URL in cached page

```
<a href="http://www.siterequest.com/apps/shopping.jsp?
shopper=AAAnb35vnM09&...."link</a>
```

Because parameter value substitution is not defined, the BIG-IP continues to serve the cached page for subsequent requests.

An example with parameter value substitution

If you configure parameter value substitution for an assembly rule, the BIG-IP changes the targeted parameter's value on the page that it serves from cache, so that the parameter you specify appears on the URL embedded in that page.

When you use parameter value substitution in assembly rules, you identify the value as a dynamic source definition in an HTTP request or number randomizer, and specify that when the BIG-IP serves the page, it embeds the named value into the appropriate location in the page's URL.

Therefore, for the example URL, when you define a parameter value substitution, the BIG-IP substitutes the original cached value for the `shopper` attribute with the value in the request.

An original request generates a cached page.

Request	Request URL	BIG-IP embedded URL in cached page
Original request	<code>http://www.siterequest.com/apps/shopping.jsp? shopper=AAAnb35vnM09&....</code>	<code><a href="http://www.siterequest.com/apps/shoppingCart.jsp? shopper=AAAnb35vnM09&...."link</code>

For subsequent requests of the example page, the BIG-IP substitutes the original cached value for the `shopper` attribute, replacing it with the value in the request.

Request	Request URL	BIG-IP embedded URL in cached page
Subsequent request	<code>http://www.siterequest.com/apps/shopping.jsp? shopper=SNkj90qcL47&....</code>	<code><a href="http://www.siterequest.com/apps/shoppingCart.jsp? shopper=SNkj90qcL47&...."link</code>

When parameter value substitution is defined, the BIG-IP substitutes the original cached value with the value from the requested page for subsequent requests, providing the ability to serve the appropriate specific content.

Accelerating Access to PDF Content

Overview: Accelerating access to PDF content with PDF linearization

Large PDF files can provide a slow response in displaying content when the entire file must download before a requested page can be accessed. The BIG-IP[®] device provides the ability to display a requested page more quickly by using PDF linearization (optimization). PDF linearization prepares the PDF file for byte serving, which enables the BIG-IP device to provide individual pages to a client when it receives byte-range requests.

All PDF files are constructed in one of two formats:

- Nonlinear. A *nonlinear* (not optimized) PDF file typically provides slower access to specific pages than a linear PDF file because a page-offset index for the document's pages is omitted. For example, PDF files that are created for high quality print output are often nonlinear.
- Linear. A *linear* (optimized) PDF file, in comparison, provides faster access to specific pages because a page-offset index for the document's pages is written at the beginning, enabling a web browser to send byte-range requests to access and display initial or specific pages before the entire file is downloaded.

When you enable PDF linearization, the BIG-IP device provides a linear PDF file, thus allowing expedient access to a requested page.

Accelerating Images with Image Optimization

Overview: Accelerating images with image optimization

You can configure *image optimization* in a BIG-IP[®] policy to reduce the size of image files, for example, by removing unnecessary metadata, by changing the format, or by increasing compression, and, consequently, accelerate the transfer of image objects across a network.

When an image object is matched to a policy node, it is modified in accordance with the acceleration rules of the policy. Configurable acceleration rules for an image object include several parameters.

Note: *Image optimization only benefits raster images. Vector images, such as SVG files, benefit little from image optimization, but can benefit from file compression. You can use file compression to improve the performance of vector images.*

Optimization of image format

An image of a supported format can be converted into any other supported format, although features of the original format that are not supported by the target format are lost upon conversion. For example, conversion of an animated GIF or multipage TIFF into a PNG only converts the first image from the original animated GIF or multipage TIFF into the PNG. Similarly, an original file loses transparency upon conversion if the target format does not support transparency.

The number of bytes after conversion typically varies from the number of bytes before conversion. Ideally, you will want to convert a file to produce a smaller file size; however, a requested conversion occurs even if the output produces a larger file size, except for conversion to PNG which only occurs if the converted file is smaller.

Typically, converting a GIF into a PNG, or converting a large PNG into a JPEG, depending upon the selected JPEG quality factor (level of compression), produces a smaller file size. Conversely, converting a supported format into a TIFF often produces an increase in file size. F5 Networks[®] recommends examination of converted file sizes for different formats to optimize performance with a reduced file size.

If the BIG-IP[®] device converts an image into a different format, it generates a correct `Content-Type` header, but it does not change the URL (which might include a file extension) in the HTML page that refers to the image.

Optimization with JPEG-XR

Application Acceleration Manager[™] now recognizes and converts images to JPEG-XR. *JPEG-XR* is an image format that offers both lossless and lossy compression with better quality per byte than JPEG. JPEG-XR is useful for compressing existing JPEG, GIF, PNG, or TIFF images. Compressed images can be significantly smaller in percentage compared to PNG or JPEG. When enabled, Application Acceleration Manager[™] will convert the images only when the request comes from a browser that supports JPEG-XR. JPEG-XR is supported natively in some browsers and in others, through plug-ins.

Browsers that support JPEG-XR:

- Internet Explorer 9+
- Internet Explorer Mobile

Optimization with WebP

Application Acceleration Manager™ now recognizes and converts images to WebP. *WebP* is an image format developed by Google that offers both lossless and lossy compression with better quality per byte than JPEG. WebP is useful for compressing existing JPEG, GIF, PNG, or TIFF images. Compressed images can be significantly smaller in percentage compared to PNG or JPEG. When enabled, Application Acceleration Manager™ will convert the images only when the request comes from a browser that supports WebP. WebP is supported natively in some browsers and in others, through plug-ins.

Browsers that support WebP:

- Chrome 9+
- Opera 12+
- Opera Mobile 11+
- Android Ice-Cream Sandwich 4.0+

Optimization with file compression

When an optimized image is a JPEG, you can set a quality level that ranges from 1 (low quality and maximum compression) through 100 (high quality and minimum compression).

Absolute compression specifies the quality level directly. A practical value for the quality level is from 30 to 100. You can perform absolute compression of a higher-quality JPEG into a lower-quality JPEG, but not the reverse.

If the original image is JPEG, and, therefore has a quality factor, *relative compression* specifies the new quality as a percentage relative to the original quality.

Optimization of headers

A JPEG image might contain an optional exchangeable image format (EXIF) header, which includes metadata, such as a date, time, author, copyright, camera model, exposure settings, global positioning coordinates, and possibly a color profile. This optional header can vary considerably in size, and does not affect display of the image (unless it contains a color profile). The EXIF header can be a significant fraction of the image; consequently, removing it can be advantageous.

Note: *If necessary, you can preserve copyright metadata when you strip other metadata from the EXIF header, by selecting the **Strip EXIF keeps copyright** check box.*

You can select one of the following options for the EXIF header.

- **Don't Strip EXIF.** The EXIF header is not changed.
- **Always Strip EXIF.** The EXIF header is always removed from the JPEG file.
- **Strip EXIF if safe.** The EXIF header is always removed, unless the header includes a color profile.
- **Apply color profile, then strip EXIF.** After the color profile is applied, the EXIF header is always removed. Use this option to convert the image to the default color profile, so that the EXIF header can be safely removed.

Optimization of sampling factor

Because human eyes perceive small changes in brightness, but not small changes in color, you can frequently use an average color for two adjacent pixels, horizontally (2x1), vertically (1x2), or both (2x2), thus improving compression with insignificant changes in quality.

You can use one of the following options to optimize the sampling factor.

- **Preserve.** The sampling factor matches the brightness and color values of the original file.
- **1x1.** Provides the same sampling factor for the brightness and color values.
- **2x1.** Averages color values for horizontal pixels.
- **1x2.** Averages color values for vertical pixels.
- **2x2.** Averages color values for vertical and horizontal pixels.

Optimization with progressive encoding

With *baseline encoding* (default), a browser renders a JPEG image from the top to the bottom as the image file downloads. However, you can sometimes improve optimization of JPEG images larger than about 10 kilobytes by using *progressive encoding*, which quickly renders a low-quality version of the entire image, and continuously improves the quality as the image file downloads. For larger image files or slow network connections, users can view rendered images faster with progressive encoding.

Optimization of color values

Reducing the number of colors in a PNG image to 256 optimally chosen color values can significantly reduce the file size with minimal degradation in the quality of the image. Because GIF images already have a maximum of 256 colors, you should not use this option when converting GIF images to PNG images.

Accelerating Video Streams with Video Delivery Optimization

About video delivery optimization

BIG-IP® *video delivery optimization* provides you with the ability to retrieve and accelerate on-demand video stream from an origin web server. The BIG-IP system sends client requests for the video stream to an origin web server, caches the response video segments, and sequentially sends optimized video responses to all authorized users.

Additionally, video delivery optimization enables you to associate video advertisements with a video stream, providing the ability to preroll advertisements, or to insert advertisements as specified by a video advertisement policy.

About caching video segments by location

You can configure BIG-IP® devices in asymmetrical deployments, symmetrical deployments, or both to optimize performance needs, positioning BIG-IP devices in accordance with higher-demand, lower-bandwidth locations within the network.

About caching popular content

A BIG-IP® device manages popular video content by evaluating several aspects, including the proximity of clients, number of requests, performance of the network, and defining values of the video segments. You can modify the resultant evaluation by changing the **Cache Priority** setting in the Responses Cached screen for a BIG-IP acceleration policy.

About video delivery optimization cache priority

You can define a caching priority level for video segments, which is useful in specifying a higher caching priority for popular video segments, by using the **Cache Priority** setting in the Responses Cached screen.

About globally configuring video delivery optimization

Optimizing video in a global network improves the video performance across significant distances. When you implement video delivery optimization in a symmetric deployment, the system caches video segments on the device closest to the client, reducing the latency and improving the quality of the video.

About video delivery optimization bit rate selection

You can specify a maximum bit rate for video delivery optimization, which limits the maximum bit rate that is available to the user. When you configure different maximum bit rates, you can designate those

specific bit rates to different types or levels of users. For example, you could create a policy node for each level of user and assign a different maximum bit rate to each node. A value of 0 indicates that the bit rate is unconstrained.

About the video Quality of Experience profile

The BIG-IP® system's video Quality of Experience (QoE) profile enables you to assess an audience's video session or overall video experience, providing an indication of customer satisfaction. The QoE profile uses static information, such as bitrate and duration of a video, and video metadata, such as URL and content type, in monitoring video streaming. Additionally, the QoE profile monitors dynamic information, which reflects the real-time network condition.

By considering both the static video parameters and the dynamic network information, the user experience can be assessed and defined in terms of a single mean opinion score (MOS) of the video session, and a level of customer satisfaction can be derived. QoE scores are logged in the `ltm` log file, located in `/var/log`, which you can evaluate as necessary.

Note that for QoE to properly process video files, the video web servers must be compliant with supported video MIME types, for example, the following MIME types.

MIME Type	Suffix
video/mp4	.f4v
video/mp4	.mp4
video/x-flv	.flv
video/x-m4v	.m4v
video/quicktime	.m4v
application/x-mpegURL	.m3u8
video/mp2t	.ts

About mean opinion score

The video Quality of Experience (QoE) profile provides a mean opinion score (MOS), derived from static and dynamic parameters associated with a video stream. The following table summarizes the resultant values.

MOS	Quality	Description
5	Excellent	Indicates a superior level of quality, with imperceptible degradation in the video stream.
4	Good	Indicates an above-average level of quality, with perceptible degradation that is acceptable.
3	Fair	Indicates an average level of quality, with perceptible degradation that detracts from the video experience.
2	Poor	Indicates a below-average level of quality, with perceptible degradation that significantly detracts from the video experience.

MOS	Quality	Description
1	Bad	Indicates a substandard level of quality, with perceptible degradation that proves to be significantly inferior and potentially unacceptable.

Compressing Content from an Origin Web Server

Overview: Enabling content compression from an origin web server

The BIG-IP[®] device can request gzip-encoded or deflate-encoded content from the origin web server to accelerate responses. When the **Enable Assembly Compression OWS** check box is selected (enabled), the BIG-IP[®] device sends an `Accept-Encoding: gzip, deflate` header to the origin web server. The origin web server complies only if it supports the compression mode; otherwise, the origin web server provides uncompressed content.

This functionality occurs independently of selecting (enabling) the **Enable Content Compression** check box, which sets the compression for the response that the BIG-IP device sends back to the client.

Accelerating Responses with Metadata Cache Responses

Overview: Using Metadata cache responses to accelerate responses

Responses from origin web servers include *entity tags* (ETags), which are arbitrary strings attached to a document that specify some characteristic of the document, such as a version, serial number, or checksum of content. A changed document includes a different ETag, enabling a client's `GET` request to use an `If-None-Match` conditional header to acquire a new copy of the document. Because not all web applications generate ETags consistently, the BIG-IP device creates its own ETag for each cached document that is based on a signature, or checksum, of the document's content. The BIG-IP device stores content signatures in the Metadata cache for other optimizations, including Intelligent Browser Referencing.

BIG-IP applications provide options to always or never send metadata. All BIG-IP applications share the same Metadata cache.

BIG-IP policies cache ETag headers, which include the following:

- Request URL
- Content signature of the response body
- Application name for the matching request
- Metadata, including the expiration time, read time, and update time for content

Advanced Metadata Cache Options for General Options

For the **General Options** list, this table describes the **Advanced** settings and strings for **Metadata Cache Options**.

Advanced control	Default	Description
Send Metadata	Always	This setting determines whether and how the BIG-IP performs Metadata caching for responses. <ul style="list-style-type: none">• Never. The BIG-IP does not cache Metadata headers.• Always. The BIG-IP always caches Metadata headers before sending them to a client.
Metadata Cache Max Size	25	Specifies the size in megabytes (MB) for the maximum Metadata cache size.

Accelerating Traffic with a Local Traffic Policy

About classifying types of HTTP traffic with a local traffic policy

An application that runs on a virtual server accelerates all HTTP traffic. You can, however, use a local traffic policy to classify types of HTTP traffic for the BIG-IP[®] system to accelerate, by specifying hosts, paths, headers, and cookies.

Important: Although you can use a local traffic policy to classify the types of HTTP traffic to accelerate, the local traffic policy overrides the **Web Acceleration** profile on the virtual server. Acceleration of HTTP traffic with the BIG-IP system should primarily be configured through a **Web Acceleration** profile, instead of a local traffic policy.

Local traffic policy matching Strategies settings

This table summarizes the strategies used for traffic policy matching.

Matching strategy	Description
First-match strategy	A <i>first-match strategy</i> executes the actions for the first rule in the Rules list that matches.
Best-match strategy	<p>A <i>best-match strategy</i> selects and executes the actions of the rule in the Rules list with the best match, as determined by the following factors.</p> <ul style="list-style-type: none">• The number of conditions and operands that match the rule.• The length of the matched value for the rule.• The priority of the operands for the rule. <p>Note: In a best-match strategy, when multiple rules match and specify an action, conflicting or otherwise, only the action of the best-match rule is executed. A best-match rule can be the lowest ordinal, the highest priority, or the first rule that matches in the Rules list.</p>
All-match strategy	<p>An <i>all-match strategy</i> executes the actions for all rules in the Rules list that match.</p> <p>Note: In an all-match strategy, when multiple rules match, but specify conflicting actions, only the action of the best-match rule is executed. A best-match rule can be the lowest ordinal, the highest priority, or the first rule that matches in the Rules list.</p>

Local traffic policy matching Requires profile settings

This table summarizes the profile settings that are required for local traffic policy matching.

Requires Setting	Description
http	Specifies that the policy matching requires an HTTP profile.
ssl	Specifies that the policy matching requires a Client SSL profile.
tcp	Specifies that the policy matching requires a TCP profile.

Local traffic policy matching Controls settings

This table summarizes the controls settings that are required for local traffic policy matching.

Controls Setting	Description
acceleration	Provides controls associated with acceleration functionality.
caching	Provides controls associated with caching functionality.
classification	Provides controls associated with classification.
compression	Provides controls associated with HTTP compression.
forwarding	Provides controls associated with forwarding functionality.
persistence	Provides controls associated with persistence functionality.
request-adaptation	Provides controls associated with request-adaptation functionality.
response-adaptation	Provides controls associated with response-adaptation functionality.
server-ssl	Provides controls associated with server-ssl functionality.

Local traffic policy matching condition types

This table summarizes the types of conditions used in policy matching.

Condition Type	Selectors and Parameters	Valid Events	Options
Client SSL	<ul style="list-style-type: none"> cipher cipher-bits protocol 	<ul style="list-style-type: none"> response request 	<ul style="list-style-type: none"> Skip this condition if it is missing from the request Use case sensitive string comparison
CPU Usage	<ul style="list-style-type: none"> 15 seconds 1 minute 5 minutes 	<ul style="list-style-type: none"> response request ssl client hello ssl server handshake 	

Condition Type	Selectors and Parameters	Valid Events	Options
		<ul style="list-style-type: none"> ssl server hello 	
Geo. IP	<ul style="list-style-type: none"> continent country code country name isp organization region code region name 	<ul style="list-style-type: none"> response request ssl client hello ssl server handshake ssl server hello 	<ul style="list-style-type: none"> Apply to traffic on remote or local side of external or internal interface. Skip this condition if it is missing from the request Use case sensitive string comparison
HTTP Basic Auth.	<ul style="list-style-type: none"> password username 		<ul style="list-style-type: none"> Skip this condition if it is missing from the request Use case sensitive string comparison
HTTP Cookie	named		<ul style="list-style-type: none"> Skip this condition if it is missing from the request Use case sensitive string comparison
HTTP Header	named	<ul style="list-style-type: none"> response request 	<ul style="list-style-type: none"> Skip this condition if it is missing from the request Use case sensitive string comparison
HTTP Host	<ul style="list-style-type: none"> host port full string 		<ul style="list-style-type: none"> Skip this condition if it is missing from the request Use case sensitive string comparison
HTTP Method	(Specify the string for the method.)		<ul style="list-style-type: none"> Skip this condition if it is missing from the request Use case sensitive string comparison
HTTP Referer	<ul style="list-style-type: none"> all extension host path path-segment port query-parameter query-string scheme unnamed-query-parameter full string 	<ul style="list-style-type: none"> request 	<ul style="list-style-type: none"> Skip this condition if it is missing from the request Use case sensitive string comparison
HTTP Set Cookie	named <ul style="list-style-type: none"> domain expiry path 		<ul style="list-style-type: none"> Skip this condition if it is missing from the request Use case sensitive string comparison

Condition Type	Selectors and Parameters	Valid Events	Options
	<ul style="list-style-type: none"> value version 		
HTTP Status	<ul style="list-style-type: none"> code text full string 		
HTTP URI	<ul style="list-style-type: none"> extension host path path-segment port query-parameter query-string scheme unnamed-query-parameter full string 		<ul style="list-style-type: none"> Skip this condition if it is missing from the request Use case sensitive string comparison
HTTP Version	<ul style="list-style-type: none"> major minor protocol full string 		<ul style="list-style-type: none"> Use case-sensitive string comparison
SSL Certificate	with index		<ul style="list-style-type: none"> Skip this condition if it is missing from the sslServerHandshake. Use case-sensitive string comparison
SSL Extension	<ul style="list-style-type: none"> alpn npn server name 	<ul style="list-style-type: none"> ssl client hello ssl server hello 	<ul style="list-style-type: none"> Skip this condition if it is missing from the sslClientHello. Use case-sensitive string comparison
TCP	<ul style="list-style-type: none"> address mss port route-domain rtt vlan vlan-id 	<ul style="list-style-type: none"> response request ssl client hello ssl server handshake ssl server hello 	<ul style="list-style-type: none"> Apply to traffic on remote or local side of external or internal interface. Use case sensitive string comparison
WebSocket	<ul style="list-style-type: none"> extension key protocol version 	<ul style="list-style-type: none"> websocket request websocket response 	<ul style="list-style-type: none"> Skip this condition if it is missing from the sslClientHello. Use case-sensitive string comparison

Local traffic policy matching Actions operands

This table summarizes the actions associated with the conditions of the rule used in policy matching.

Target	Type	Valid Events	Action
acceleration	string/number	<ul style="list-style-type: none"> request 	<ul style="list-style-type: none"> disable enable
cache	string	<ul style="list-style-type: none"> request response 	<ul style="list-style-type: none"> disable enable pin true
compress	string	<ul style="list-style-type: none"> request response 	<ul style="list-style-type: none"> disable enable
decompress	string	<ul style="list-style-type: none"> request response 	<ul style="list-style-type: none"> disable enable
forward	string	<ul style="list-style-type: none"> request 	<ul style="list-style-type: none"> reset select clone-pool member nexthop node pool rateclass snat snatpool vlan vlan-id
http-cookie	string	<ul style="list-style-type: none"> request 	<ul style="list-style-type: none"> insert name (required) value (required) <hr/> <p><i>Note: This parameter supports Tcl expressions.</i></p> <hr/> <ul style="list-style-type: none"> remove name (required)
http-header	string/number	<ul style="list-style-type: none"> request response 	<ul style="list-style-type: none"> insert name (required) value (required) <hr/> <p><i>Note: This parameter supports Tcl expressions.</i></p> <hr/>

Target	Type	Valid Events	Action
			<ul style="list-style-type: none"> • remove • name (required) • replace • name (required) • value (required) <hr/> <p><i>Note: This parameter supports Tcl expressions.</i></p> <hr/>
http-host	string	<ul style="list-style-type: none"> • request 	<ul style="list-style-type: none"> • replace • value <hr/> <p><i>Note: This parameter supports Tcl expressions.</i></p> <hr/>
http-referer	string	<ul style="list-style-type: none"> • request 	<ul style="list-style-type: none"> • insert • value (required) <hr/> <p><i>Note: This parameter supports Tcl expressions.</i></p> <hr/>
			<ul style="list-style-type: none"> • remove • replace • value <hr/> <p><i>Note: This parameter supports Tcl expressions.</i></p> <hr/>
http-reply	string	<ul style="list-style-type: none"> • request • response 	<ul style="list-style-type: none"> • redirect • location (required) <hr/> <p><i>Note: This parameter supports Tcl expressions.</i></p> <hr/>
http-set-cookie	string/number	<ul style="list-style-type: none"> • response 	<ul style="list-style-type: none"> • insert • name (required) • domain <hr/> <p><i>Note: This parameter supports Tcl expressions.</i></p> <hr/>
			<ul style="list-style-type: none"> • path

Target	Type	Valid Events	Action
http-uri	string/number	• response	<i>Note: This parameter supports Tcl expressions.</i>
			• value (required)
			<i>Note: This parameter supports Tcl expressions.</i>
			• remove • name (required)
http-uri	string/number	• response	• replace • path
			<i>Note: This parameter supports Tcl expressions.</i>
			• query-string
			<i>Note: This parameter supports Tcl expressions.</i>
http-uri	string/number	• response	• value
			<i>Note: This parameter supports Tcl expressions.</i>
log	string/number	• request • response	• write • message (required)
			<i>Note: This parameter supports Tcl expressions.</i>
pem	string/number	• request • response	• classify • application • category • defer • protocol
persist	string/number	• request • response	• disable • source-address • carp
			<i>Note: This parameter supports Tcl expressions.</i>
			• cookie-hash

Target	Type	Valid Events	Action
			<ul style="list-style-type: none"> • cookie-insert • cookie-passive • cookie-rewrite • destination-address • disable • hash <hr/> <p><i>Note: This parameter supports Tcl expressions.</i></p> <hr/> <ul style="list-style-type: none"> • source-address • universal <hr/> <p><i>Note: This parameter supports Tcl expressions.</i></p> <hr/>
request-adapt	string/number	<ul style="list-style-type: none"> • request • response 	<ul style="list-style-type: none"> • disable • enable
response-adapt	string/number	<ul style="list-style-type: none"> • request • response 	<ul style="list-style-type: none"> • disable • enable
server-ssl	string/number	<ul style="list-style-type: none"> • request 	<ul style="list-style-type: none"> • disable • enable
tcl	string/number	<ul style="list-style-type: none"> • request • response 	<ul style="list-style-type: none"> • set-variable • name (required) • expression (required) <hr/> <p><i>Note: This parameter supports Tcl expressions.</i></p> <hr/>
tcp-nagle	string/number	<ul style="list-style-type: none"> • request 	<ul style="list-style-type: none"> • disable • enable

Accelerating Traffic with Intelligent Client Cache

About intelligent client cache

Intelligent Client Cache (ICC) is a web acceleration technique for mobile and desktop browsers that support HTML5. ICC uses HTML5 local storage to build a cache of documents and resources. It does this either by replacing the link to CSS/JavaScript/Image by inlining them into the HTML document, or by replacing the link to CSS/JavaScript/Image by adding reference to content that might already be in the client's local storage. *Local storage* is a simple key-value storage in HTML5 and is shared across all browser windows and tabs. Most common implementations allow 5-10 MB per domain. Client-side JavaScript code tracks the resources cached and interacts with the server-side code to ensure that only changed resources are downloaded on subsequent requests.

Browsers that support web storage:

- Internet Explorer version 8+
- FireFox version 3.6+
- Opera 10.5+
- Chrome 5+
- Safari 4+
- iOS 3.2+
- Android 2.1+

Using Forward Error Correction to Mitigate Packet Loss

Overview: Using forward error correction (FEC) to mitigate packet loss

The BIG-IP[®] system performs forward error correction (FEC) by adding redundancy to the transmitted information. FEC provides a loss correction facility for all IP-based protocols optimized by Application Acceleration Manager[™]. All iSession[™] traffic can benefit from FEC loss mitigation, which is preferred over aggressive TCP retransmission in shared network environments.

To implement forward error correction, the BIG-IP system aggregates packets for a specified amount of time, divides the load into the specified number of equal packets (source packets), and adds the specified number of redundant (repair) packets. With adaptive FEC, the system adjusts these numbers as it measures the link error rate.

If you are configuring FEC on a central BIG-IP device for a server that does not initiate traffic, you can configure a FEC tunnel with an undefined remote address. You then configure a separate FEC tunnel from each remote BIG-IP device that handles client-initiated traffic to the central BIG-IP device. You can also configure a FEC tunnel between the local BIG-IP device and any other BIG-IP device that has a FEC tunnel with an undefined remote address.

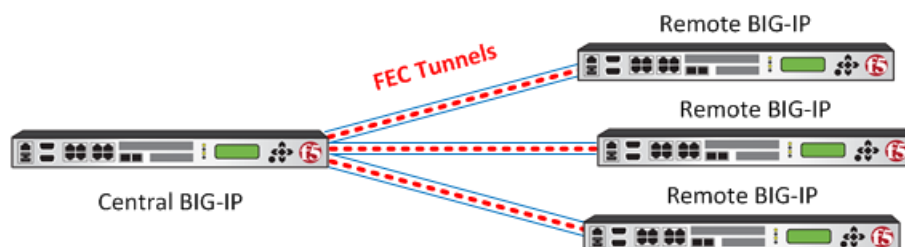


Figure 21: FEC configuration between BIG-IP devices

In addition to configuring FEC between two BIG-IP systems, you can configure FEC between an edge client and a BIG-IP system that has Access Policy Manager[®] licensed. Consult the Access Policy Manager (APM[®]) documentation for information about configuring the client access deployment.

Note: Before you can configure forward error correction (FEC), you must have licensed and provisioned Application Acceleration Manager (AAM[™]).

About forward error correction (FEC)

Forward error correction (FEC) is an acceleration technique for all kinds of traffic, including TCP and UDP traffic on lossy networks. FEC controls data transmission errors over unreliable or noisy communication channels. With FEC, the sender encodes messages with an extra error-correcting code (ECC). The redundancy allows the receiver to detect a limited number of errors that might occur anywhere in the message, and often to correct these errors without retransmission.

Packet loss occurs when one or more packets traveling across a network fail to reach their destination. Packet loss can be caused by a number of factors that inevitably result in highly noticeable performance issues, particularly with realtime protocols, streaming technologies, voice-over-IP, online gaming, and video

conferencing. Some network transport protocols, such as TCP, provide for reliable delivery of packets. In the event of packet loss, the receiver might ask for retransmission, or the sender automatically resends any segments that have not been acknowledged. Although TCP can recover from packet loss, retransmitting missing packets causes the overall throughput of the connection to decrease. Error correction occurs without the need for a reverse channel to request retransmission of data, but at the cost of a fixed, higher forward channel bandwidth. Therefore, FEC is most useful in situations where retransmissions are costly or impossible.

Using the Request Logging Profile

Overview: Configuring a Request Logging profile

The Request Logging profile gives you the ability to configure data within a log file for HTTP requests and responses, in accordance with specified parameters.

Task summary

Perform these tasks to log HTTP request and response data.

About the Request Logging profile

Many sites perform traffic analysis against the HTTP log files that their web servers generate. With the Request Logging profile, you can specify the data and the format for HTTP requests and responses that you want to include in a log file. If you prefer, you can tailor the information that appears in the logs so that the logs work seamlessly with whatever analysis tools you use for your origin web server's HTTP log files.

Standard log formats

Log headers appear in the lines at the top of a log file. You can use log headers to identify the type and order of the information written to each line in the log file. Some log analysis software also uses log headers to determine how to parse a log file.

There are three common conventions for log headers shown here.

Convention	Description
No header line	Apache™ web servers use this option. By default, Apache web servers write access logs in a format that is identical to the NCSA Common format.
NCSA Common or Combined headers	Netscape® servers, and their descendants (such as the iPlanet™ Enterprise Server) write a log header line that is unique to this family of servers. These servers generally use either the NCSA Common or Combined log format, and the log header lines are composed of keywords. For example: <pre>#format=%Ses->client.ip% - %Req->vars.auth-user% [%SYSDATE%]</pre>
W3C headers	Most Microsoft® Internet Information Services (IIS) web servers write log files in the extended log file format, which is defined by a W3C working draft.

The logging information that is commonly used by origin web servers consists of the following conventions:

- NCSA Common (no log header)
- NCSA Common (Netscape log header)
- NCSA Combined (no log header)
- NCSA Combined (Netscape log header)

- W3C Extended

NCSA Common log format example

This is the NCSA Common log format syntax:

```
host rfc931 username [date:time UTC_offset]
"method URI?query_parameters protocol" status bytes
```

Here is an example that uses this syntax:

```
125.125.125.2 - - [03/Apr/2011:23:44:03 -0600]
"GET /apps/example.jsp?sessionID=34h76 HTTP/1.1" 200 3045
```

NCSA Combined log format example

This is the NCSA Combined log format syntax:

```
host rfc931 username [date:time UTC_offset]
"method URI?query_parameters protocol" status bytes
"referrer" "user_agent" "cookie"
```

Here is an example that uses this syntax:

```
125.125.125.2 - - [03/Apr/2011:23:44:03 -0600]
"GET /apps/example.jsp?sessionID=34h76 HTTP/1.1" 200 3045
"http://www.siterequest.com" "Mozilla/4.0 (compatible; MSIE 6.0; Windows NT
5.0)"
"UserID=ssarette;Category=PDA;Selection=Various"
```

W3C Extended log format example

This is the W3C extended log format syntax:

```
date time rfc931 username host method URI query_parameters
status bytes request_length time_taken protocol user_agent cookie referrer
```

Following is an example that uses this syntax:

```
2011-04-03 23:44:03 205.47.62.112 - 125.125.125.2
GET /apps/example.jsp sessionID=34h76 200 3045 124 138
HTTP/1.1 Mozilla/4.0+(compatible;+MSIE+6.0;+Windows+NT+5.0
UserID=ssarette;Category=PDA;Selection=Various http://www.siterequest.com
```


Request Logging profile settings

With the Request Logging profile, you can specify the data and the format for HTTP requests and responses that you want to include in a log file.

General Properties

Setting	Value	Description
Name	No default	Specifies the name of the profile.
Parent Profile	Selected predefined or user-defined profile	Specifies the selected predefined or user-defined profile.

Request Settings

Setting	Value	Description
Request Logging	Disabled	Enables logging for requests.
Template		Specifies the directives and entries to be logged.
HSL Protocol	UDP	Specifies the protocol to be used for high-speed logging of requests.
Pool Name	None	Defines the pool associated with the virtual server that is logged.
Respond On Error	Disabled	Enables the ability to respond when an error occurs.
Error Response	None	Specifies the response text to be used when an error occurs. For example, the following response text provides content for a 503 error. <pre><html> <head> <title>ERROR</title> </head> <body> <p>503 ERROR-Service Unavailable</p> </body> </html></pre>
Close On Error	Disabled	When enabled, and logging fails, drops the request and closes the connection.
Log Logging Errors	Disabled	Enables the ability to log any errors when logging requests.
Error Template	None	Defines the format for requests in an error log.
HSL Error Protocol	UDP	Defines the protocol to be used for high-speed logging of request errors.
Error Pool Name	None	Specifies the name of the error logging pool for requests.

Response Settings

Setting	Value	Description
Response Logging	Disabled	Enables logging for responses.
Log By Default	Enabled	Defines whether to log the specified settings for responses by default.
Template	None	Specifies the directives and entries to be logged.
HSL Protocol	UDP	Specifies the protocol to be used for high-speed logging of responses.
Pool Name	None	Defines the pool name associated with the virtual server that is logged.
Log Logging Errors	Disabled	Enables the ability to log any errors when logging responses.
Error Template	None	Defines the format for responses in an error log.
HSL Error Protocol	UDP	Defines the protocol to be used for high-speed logging of response errors.
Error Pool Name	None	Specifies the name of the error logging pool for responses.

Request Logging parameters

This table lists all available parameters from which you can create a custom HTTP Request Logging profile. These are used to specify entries for the **Template** and **Error Template** settings. For each parameter, the system writes to the log the information described in the right column.

Table 9: Request logging parameters

Parameter	Log file entry description
BIGIP_BLADE_ID	An entry for the slot number of the blade that handled the request.
BIGIP_CACHED	An entry of <code>Cached status: true</code> , if the response came from BIG-IP® cache, or <code>Cached status: false</code> , if the response came from the server.
BIGIP_HOSTNAME	An entry for the configured host name of the unit or chassis.
CLIENT_IP	An entry for the IP address of a client, for example, 192.168.74.164.
CLIENT_PORT	An entry for the port of a client, for example, 80.
DATE_D	A two-character entry for the day of the month, ranging from 1 (note the leading space) through 31.
DATE_DAY	An entry that spells out the name of the day.
DATE_DD	A two-digit entry for the day of the month, ranging from 01 through 31.
DATE_DY	A three-letter entry for the day, for example, Mon.
DATE_HTTP	A date and time entry in an HTTP format, for example, Tue, 5 Apr 2011 02:15:31 GMT.
DATE_MM	A two-digit month entry, ranging from 01 through 12.

Parameter	Log file entry description
DATE_MON	A three-letter abbreviation for a month entry, for example, APR.
DATE_MONTH	An entry that spells out the name of the month.
DATE_NCSA	A date and time entry in an NCSA format, for example, dd/mm/yy:hh:mm:ss ZNE.
DATE_YY	A two-digit year entry, ranging from 00 through 99.
DATE_YYYY	A four-digit year entry.
HTTP_CLASS	The name of the <code>httpclass</code> profile that matched the request, or an empty entry if a profile name is not associated with the request.
HTTP_KEEPALIVE	A flag summarizing the HTTP1.1 keep-alive status for the request: <code>ay</code> if the HTTP1.1 keep-alive header was sent, or an empty entry if not.
HTTP_METHOD	An entry that defines the HTTP method, for example, GET, PUT, HEAD, POST, DELETE, TRACE, or CONNECT.
HTTP_PATH	An entry that defines the HTTP path.
HTTP_QUERY	The text following the first <code>?</code> in the URI.
HTTP_REQUEST	The complete text of the request, for example, <code>\$METHOD \$URI \$VERSION</code> .
HTTP_STATCODE	The numerical response status code, that is, the status response code excluding subsequent text.
HTTP_STATUS	The complete status response, that is, the number appended with any subsequent text.
HTTP_URI	An entry for the URI of the request.
HTTP_VERSION	An entry that defines the HTTP version.
NCSA_COMBINED	An NCSA Combined formatted log string, for example, <code>\$NCSA_COMMON \$Referer \${User-agent} \$Cookie</code> .
NCSA_COMMON	An NCSA Common formatted log string, for example, <code>\$CLIENT_IP - - \$DATE_NCSA \$HTTP_REQUEST \$HTTP_STATCODE \$RESPONSE_SIZE</code> .
RESPONSE_MSECS	The elapsed time in milliseconds (ms) between receiving the request and sending the response.
RESPONSE_SIZE	An entry for the size of response in bytes.
RESPONSE_USECS	The elapsed time in microseconds (μs) between receiving the request and sending the response.
SERVER_IP	An entry for the IP address of a server, for example, <code>10.10.0.1</code> .
SERVER_PORT	An entry for the port of a logging server.
SNAT_IP	An entry for the self IP address of the BIG-IP-originated connection to the server when SNAT is enabled, or an entry for the client IP address when SNAT is not enabled.
SNAT_PORT	An entry for the port of the BIG-IP-originated connection to the server when SNAT is enabled, or an entry for the client port when SNAT is not enabled.
TIME_AMPM	A twelve-hour request-time qualifier, for example, AM or PM.
TIME_H12	A compact twelve-hour time entry for request-time hours, ranging from 1 through 12.
TIME_HRS	A twelve-hour time entry for hours, for example, <code>12 AM</code> .

Parameter	Log file entry description
TIME_HH12	A twelve hour entry for request-time hours, ranging from 01 through 12.
TIME_HMS	An entry for a compact request time of H:M:S, for example, 12:10:49.
TIME_HH24	A twenty-four hour entry for request-time hours, ranging from 00 through 23.
TIME_MM	A two-digit entry for minutes, ranging from 00 through 59.
TIME_MSECS	An entry for the request-time fraction in milliseconds (ms).
TIME_OFFSET	An entry for the time zone, offset in hours from GMT, for example, -11.
TIME_SS	A two-digit entry for seconds, ranging from 00 through 59.
TIME_UNIX	A UNIX time entry for the number of seconds since the UNIX epoch, for example, 00:00:00 UTC, January 1st, 1970.
TIME_USECS	An entry for the request-time fraction in microseconds (μ s).
TIME_ZONE	An entry for the current Olson database or tz database three-character time zone, for example, PDT.
VIRTUAL_IP	An entry for the IP address of a virtual server, for example, 192.168.10.1.
VIRTUAL_NAME	An entry for the name of a virtual server.
VIRTUAL_POOL_NAME	An entry for the name of the pool containing the responding server.
VIRTUAL_PORT	An entry for the port of a virtual server, for example, 80.
VIRTUAL_SNATPOOL_NAME	The name of the Secure Network Address Translation pool associated with the virtual server.
WAM_APPLICATION_NAM	An entry that defines the name of the BIG-IP [®] acceleration application that processed the request.
WAM_X_WA_INFO	An entry that specifies a diagnostic string (X-WA-Info header) used by BIG-IP acceleration to process the request.
NULL	Undelineated strings return the value of the respective header.

Monitoring BIG-IP Acceleration Application Performance

Overview: Monitoring the performance of a BIG-IP acceleration application

The BIG-IP's performance reports provide information about page requests, the frequency of those requests, and how well the BIG-IP system serviced those requests from cache. Additionally, performance reports provide information about the acceleration application, policy, policy node, HTTP response status, S-code, size range of the response, response object type, and ID of the BIG-IP system or browser making the request.

The BIG-IP system provides three types of performance reports.

- **Traffic Reports.** These reports display the number of requests (hits) received, and responses served, by the BIG-IP system.
- **Byte Reports.** These reports display the bytes of content that the BIG-IP system has sent in response to requests.
- **Response Reports.** These reports display the average amount of time it takes the BIG-IP system to respond to a request from the client.

You can use these performance reports to evaluate your acceleration policies, adjusting them as required to maximize client access to your applications. The individual performance reports display content according to the persistent parameters that you select for the filter. You can also save performance reports to a specified file type so that you can import them into specific applications.

***Note:** Enabling performance monitoring for a BIG-IP acceleration application can degrade overall performance and should only be used temporarily.*

Advanced performance monitor settings for general options

For the **General Options** list, this table describes the **Advanced** settings and strings for **Performance Monitor Options**.

Advanced control	Default	Description
Performance Monitor	Disabled	Specifies whether performance monitoring for this application is enabled. Enabling performance monitoring on many applications might affect the overall performance of the BIG-IP.
Data Retention Period	30	Specifies the period, in days, that the performance data must be preserved.

Overview: ROI reports

You can evaluate the benefits and performance improvements of acceleration functionality for an application by examining ROI statistics. Acceleration ROI Statistics provide data on compression, caching, minification, inlining, image optimization, and Intelligent Browser Referencing (IBR), which you can assess to determine the current acceleration performance, and, based on those results, refine the acceleration performance.

About Byte Savings reports

Byte Savings reports provide statistics for cached objects that are either optimized or simply cached, as a means to minimize the traffic sent to a client. Byte Savings reports include four types of reports: Caching Bytes Saved, Compression Bytes Saved, Image Optimization Bytes Saved, and Minification Bytes Saved.

Caching Bytes Saved reports

Cache Bytes Saved reports provide statistics about the number of kilobytes served from cache, describing the reduction in bandwidth between the origin web server and the BIG-IP® system.

Compression Bytes Saved reports

Compression Bytes Saved reports provide statistics about the reduction in kilobytes sent to the client, when BIG-IP system performs compression instead of the origin web server.

Image Optimization Bytes Saved reports

Image Optimization Bytes Saved reports provide statistics about optimized images in kilobytes, which reduces the sizes of images sent to the client.

Minification Bytes Saved reports

Minification Bytes Saved reports provide statistics about minified JavaScript and cascading style sheet (CSS) objects in kilobytes, reducing the sizes of objects sent to the client.

About Caching Requests Saved reports

Caching Requests Saved reports provide statistics about the number of requests that are served from cache, which reduces the load for the origin web server.

About IBR Savings reports

IBR Savings reports provide statistics about Intelligent Browser Referencing (IBR) links that have been sent or received from a client. IBR Saving reports include two types of reports: Client IBR'd Links and Client IBR'd Links Received reports.

Client IBR'd Links reports

Client IBR'd Links reports provide statistics about the number of IBR links that have been sent to a client. You will want to use the statistics for this setting in combination with the Client IBR'd Links Received statistics, to determine the number of links that eliminated unnecessary conditional GET requests by increasing the content expire time.

Client IBR'd Links Received

Client IBR'd Links Received reports provide statistics about the number of IBR links that have been received from a client. You will want to use the statistics for this setting in combination with the Client IBR'd Links statistics, to determine the number of links that eliminated unnecessary conditional GET requests by increasing the content expire time.

About Inlined Links reports

Inlined Links reports provide statistics about the number of links that become inlined when sending a response to a client, reducing the number of GET requests from the client.

About ICC Savings reports

ICC Savings reports provide statistics about links that Intelligent Client Cache (ICC) functionality provides for a client. ICC Savings reports include two types of reports: ICC Inlined Links and ICC Referenced Links reports.

ICC Inlined Links

ICC Inlined Links reports provide statistics about the number of links that become inlined for a client by ICC functionality, reducing the number of GET requests from the client.

ICC Referenced Links

ICC Referenced Links reports provide statistics about the number of links referencing content that might exist in the client's local storage, added by ICC functionality.

Managing Deduplication

What is symmetric data deduplication?

Application Acceleration Manager™ (AAM™) uses *symmetric data deduplication* to reduce the amount of bandwidth consumed across a WAN link for repeated data transfers. This feature is available only with an Application Acceleration Manager™ (AAM™) license.

With data deduplication, the system performs pattern matching on the transmitted WAN data, rather than caching. If any part of the transmitted data has already been sent, BIG-IP® system replaces the previously transmitted data with references. As data flows through the pair, each device records the byte patterns and builds a synchronized dictionary. If an identical pattern of bytes traverses the WAN more than once, the BIG-IP closest to the sender replaces the byte pattern with a reference to it, compressing the data. When the reference reaches the other side of the WAN, the remote BIG-IP device replaces the reference with the data, restoring the data to its original format.

Which codec do I choose?

Symmetric data deduplication (SDD) offers two versions, called codecs. *SDD v3* is appropriate for most AAM™ installations, particularly in large networks, such as hub and spoke, or mesh deployments. *SSD v2* is an alternative for installations with fewer than eight high-speed links, such as for data replication between data centers.

For deduplication to occur, the same codec must be selected on both iSession endpoints. If the selected codecs do not match, deduplication does not occur, although other symmetric optimization features, such as compression, still take place.

About Discovery

About discovery on BIG-IP AAM systems

To simplify configuration, particularly in large networks, BIG-IP® systems licensed and provisioned for acceleration perform two types of discovery.

- *Dynamic discovery* of remote endpoints occurs when the local BIG-IP system detects a remote iSession endpoint on the other side of the WAN.
- *Local subnet discovery* occurs, for example, when a client request to a server triggers the server-side BIG-IP device to discover and display the subnet that is connected to the server.

About subnet discovery

An *advertised route* is a subnet that can be reached through a iSession™ connection. After the iSession connection is configured between two BIG-IP® devices, they automatically exchange advertised route specifications between the endpoints. The local endpoint needs to advertise the subnets to which it is connected so that the remote endpoint can determine the destination addresses for which traffic can be optimized. Advertised routes configured on the local endpoint become remote advertised routes on the remote endpoint; that is, the BIG-IP device on the other side of the WAN.

When a BIG-IP device is deployed in a large scale network with large number of servers, and many of them belong to different subnets, manually configuring local optimization subnets can be very time consuming. Subnet Discovery is designed to ease such configuration challenges. With local subnet discovery, instead of requiring manual configuration of local subnets for traffic optimization, the BIG-IP system automatically discovers the local optimization subnet when traffic flows from a BIG-IP device on one side of the WAN to a BIG-IP device on the other side.

Note: A TCP request from the client to the server is the action that triggers discovery, not a ping between two endpoints.

About dynamic discovery of remote endpoints

Dynamic discovery is a process through which the BIG-IP® system identifies and adds remote endpoints automatically. The process occurs when the BIG-IP device receives traffic that is matched by a virtual server with an iSession™ profile, but does not recognize the remote destination. When a BIG-IP device receives a request destined for a location on the network behind the BIG-IP device on the other side of the WAN, the first BIG-IP device sends out TCP options or ICMP probes to discover, authenticate, and initiate communication with the new remote endpoint.

Note: A TCP request from the client to the server is the action that triggers discovery, not a ping between two endpoints.

Legal Notices

Legal notices

Publication Date

This document was published on November 6, 2017.

Publication Number

MAN-0467-04

Copyright

Copyright © 2017, F5 Networks, Inc. All rights reserved.

F5 Networks, Inc. (F5) believes the information it furnishes to be accurate and reliable. However, F5 assumes no responsibility for the use of this information, nor any infringement of patents or other rights of third parties which may result from its use. No license is granted by implication or otherwise under any patent, copyright, or other intellectual property right of F5 except as specifically described by applicable user licenses. F5 reserves the right to change specifications at any time without notice.

Trademarks

For a current list of F5 trademarks and service marks, see <http://www.f5.com/about/guidelines-policies/trademarks/>.

All other product and company names herein may be trademarks of their respective owners.

Patents

This product may be protected by one or more patents indicated at: <https://f5.com/about-us/policies/patents>

Export Regulation Notice

This product may include cryptographic software. Under the Export Administration Act, the United States government may consider it a criminal offense to export this product from the United States.

RF Interference Warning

This is a Class A product. In a domestic environment this product may cause radio interference, in which case the user may be required to take adequate measures.

FCC Compliance

This equipment has been tested and found to comply with the limits for a Class A digital device pursuant to Part 15 of FCC rules. These limits are designed to provide reasonable protection against harmful interference when the equipment is operated in a commercial environment. This unit generates, uses, and can radiate radio frequency energy and, if not installed and used in accordance with the instruction manual, may cause harmful interference to radio communications. Operation of this equipment in a residential area is likely to cause harmful interference, in which case the user, at his own expense, will be required to take whatever measures may be required to correct the interference.

Any modifications to this device, unless expressly approved by the manufacturer, can void the user's authority to operate this equipment under part 15 of the FCC rules.

Canadian Regulatory Compliance

This Class A digital apparatus complies with Canadian ICES-003.

Standards Compliance

This product conforms to the IEC, European Union, ANSI/UL and Canadian CSA standards applicable to Information Technology products at the time of manufacture.

Index

A

- acceleration
 - introduction 11
- acceleration policies
 - about 47
 - creating user-defined 49
 - customization 49
 - exported to XML files 50
 - inheritance rule parameters 56
 - inheritance rule parameters override 57
 - Policy Tree modification 59
 - publication 49
 - rule inheritance 55
 - screen access 47
 - types 48
- acceleration policy
 - selection 48
- Acceleration Policy Editor role
 - BIG-IP access 50
- actions operands settings
 - for local traffic policy matching 143
- advertised routes
 - description 163
- Application Acceleration Manager
 - enabling 19
- applications
 - management 14
 - monitoring 14
- applications advanced settings
 - performance monitor options 157
- Applications advanced settings
 - Debug Options 78
 - Metadata Cache Options 137
- Applications screen Advanced settings
 - IBR options 114
- assembly rules
 - management of content served from origin web servers 83
 - overview 83

B

- bandwidth controllers
 - compared with rate shaping 37
- bandwidth control policies
 - and SNMP 37
 - dynamic, about 38
 - dynamic, example of 39
 - overview 37
 - static, about 37
- base rate 42
- base throughput rate 42
- BIG-IP acceleration policies
 - options 48
- BIG-IP Application Acceleration
 - introduction 14

- burst reservoir
 - depleting 43
 - replenishing 43
- burst size 42–43
- Byte Savings reports
 - about 158

C

- Cache-Control headers
 - management 68
 - max-age directives 69
 - no-cache directives 68
- cached content
 - serving when web server is unavailable 89
- Cache on first hit setting
 - using 99
- caching
 - about 13
- Caching Bytes Saved reports
 - about 158
- Caching Requests Saved reports 158
- cascading style sheet files
 - about inlining 115
 - about minification 115
 - about optimization 115
 - about reordering 115
 - inlining 116
 - reordering 115
- Cascading Style Sheet files
 - about concatenation 116
- ceiling rate 42
- CIFS optimization
 - about 25
- Client IBR'd Links Received reports
 - about 158
- Client IBR'd Links reports
 - about 158
- codec
 - choosing for deduplication 161
- color values
 - minimizing 129
- compression
 - configuring for symmetric optimization 24
 - enabling from origin web server 135
- Compression Bytes Saved reports
 - about 158
- concatenation
 - and Cascading Style Sheet files 116
 - and JavaScript files 116
- connection pooling
 - with OneConnect 33
- CSS files
- See also Cascading Style Sheet files
 - about inlining 115
 - about minification 115
 - about optimization 115
 - about reordering 115

CSS files (*continued*)
 inlining 116
 reordering 115
 See also Cascading Style Sheet files
custom cache-control directives
 using 89

D

data centers
 about 12
data compression
 about 12
data deduplication
 about 12
deduplication
 choosing a codec 161
 described 161
discovery
 and advertised routes 163
 description 163
 of local subnets 163
 of remote endpoints 163
distributed BIG-IP application acceleration
 deployment 14
DNS prefetching 116
drop policy 46
dynamic bandwidth control policies, See bandwidth control policies
dynamic discovery
 for BIG-IP systems 163

E

ETag
 including in metadata 137

F

FEC, See forward error correction (FEC)
forward error correction (FEC)
 about 149
 overview 149

H

header format
 optimizing 128
HTML
 about inlining 115
 about minification 115
 about optimization 115
 about reordering 115
HTML files
 about minification 115
HTTP/2 profile
 about 29, 103
 overview 29, 103
HTTP/2 profile settings
 defined 30, 104
 listed 30, 104

HTTP2 profile
 overview 28, 103
HTTP Compression profile
 about 19
 options 19
HTTP data type
 regular expression strings 74
 request parameters 72
HTTP data type parameters
 specification for a rule 62–63
 specification of client IP 66
 specification of content type 66
 specification of cookie 64
 specification of extension 63
 specification of header 65
 specification of host 62
 specification of method 65
 specification of path 62
 specification of path segment 64
 specification of protocol 65
 specification of query parameter 63
 specification of referrer 65
 specification of user agent 65
HTTP protocol
 optimization 13
HTTP request
 about process 60
 using parameters 59
HTTP request latency
 minimizing 31
HTTP request logging
 about 151
 and code elements 154
 and profile settings 153
HTTP request logging profile, overview 151
HTTP request queuing
 overview 101
HTTP requests
 configuration of rules 61
 requirements for servicing 59
HTTP response
 about process 61
HTTP responses
 configuration of rules 66
 requirements for caching 61
HTTP traffic
 managing with HTTP2 profile 28
 managing with SPDY profile 107

I

IBR
 about adaptive lifetime 112
 about conditional GET requests 111
 for CSS 112
 for HTML 111
IBR Savings reports
 about 158
ICC 147
ICC Inlined Links reports
 about 159

- ICC Referenced Links reports
 - about 159
- ICC Savings reports
 - about 159
- image format
 - optimizing 127–128
- image optimization
 - for color values 129
 - for format 127–128
 - for headers 128
 - for progressive encoding 129
 - for sampling factor 129
 - overview 127
- Image Optimization Bytes Saved reports
 - about 158
- images
 - optimizing 127–128
- Inlined Links reports
 - about 159
- inlining
 - and cascading style sheet files 115
 - and CSS files 115
 - and JavaScript files 115
 - cascading style sheet files 116
 - CSS files 116
 - JavaScript files 116
- Intelligent Browser Referencing
 - about adaptive lifetime 112
 - about conditional GET requests 111
 - example 113
 - for CSS 112
 - for HTML 111
 - overview 111
 - task summary 72
- Intelligent Client Cache
 - about 147
- invalidation
 - for an application 91
- invalidations rules
 - cached content 93
 - lifetime 92
 - overview 91
 - parameters 93
 - request header matching criteria 93
 - triggers 91
- iSession
 - and symmetric optimization 17
- iSession profiles
 - about 24
 - modifying compression 24

J

- JavaScript files
 - about concatenation 116
 - about inlining 115
 - about minification 115
 - about optimization 115
 - about reordering 115
 - inlining 116
 - reordering 115
- JPEG-XR 127

- JS files, See JavaScript files

L

- LAN traffic optimization
 - and TCP protocol 26
- latency
 - minimizing 31
- lifetime managed responses
 - about 87
- lifetime rules
 - about 87
- load balancing on servers
 - about 11
- local traffic policy
 - accelerating traffic 139
 - using to classify types of HTTP traffic 139
- local traffic policy matching
 - actions operands settings 143
 - controls settings 140
 - operands settings 140
 - requires settings 140
 - strategies settings 139

M

- managed requests
 - about 87
- MAPI optimization
 - about 25
- max age
 - about 88
 - compiled responses value 76
- mean opinion score, See MOS
- meta characters
 - pattern matching 21, 76
- Metadata responses
 - about using 137
- meta tags
 - using on rules 68
- minification
 - and HTML 115
 - of cascading style sheet files 115
 - of CSS files 115
 - of HTML 115
 - of JavaScript files 115
- Minification Bytes Saved reports
 - about 158
- monitoring performance
 - about 157
- MOS
 - and Video Quality of Experience 132
- MPTCP
 - and mobile traffic optimization 27
- mptcp-mobile-optimized profile
 - about settings 27
- MultiConnect
 - example 120
 - overview 119

N

no-cache directive
inserting into header 90

NTLM

and OneConnect 35

NTLM profile type

defined 32

O

object

classification 95

object type

classification 95

classification by group 95

object types

management 95

OneConnect

and NTLM 35

OneConnect profile type

defined 33

optimization

of cascading style sheet files 115

of CSS files 115

of JavaScript files 115

optimized images

accelerating 127–128

origin web server directives

preserving to downstream devices 89

origin web server headers

preserving to downstream devices 89

P

packet loss

mitigating with FEC 149

parameters

for HTTP request logging 154

for request logging 154

parameter value substitution

about number randomizer 122

about parameters 121

example 122

serving specific content 121

parent class 45

passwords

and NTLM profiles 32

pattern matching

meta characters 21, 76

PDF linearization

overview 125

performance monitoring

about 157

Policy Editor screen

overview 53

parts 54

viewing Policy Tree 55

policy matching

configuration example 53

controls settings 140

overview 50

policy matching (*continued*)

requires settings 140

resolution rules when multiple nodes match 50

policy matching resolution rules

exact path match 51

multiple extension matches 51

single extension node match 51

single path segment match 51

unmatched requests 52

prefetching, See DNS prefetching

profile

about HTTP request logging 151

about Request Logging 151

profiles

about HTTP Compression 19

about iSession 24

about MAPI 25

about TCP 26

about Web Acceleration 19

Web Acceleration settings 20

profile settings

for HTTP/2 30, 104

profile types

for HTTP/2 29, 103

progressive encoding

optimizing 129

proxying rules

descriptions of parameters 85

overview 85

Q

QoE, See video Quality of Experience

queue method setting

using priority FIFO 45

using stochastic fair queue 45

R

rate class

about 41

and queue method 45

rate class name 42

rate shaping

and base rate 42

and burst size 42–43

and ceiling rate 42

and direction setting 44

and parent class 45

and policy 45

compared with bandwidth controllers 37

definition 41

regular expressions

using on rules 68

remote endpoints

about discovery of 163

reordering

cascading style sheet files 115

CSS files 115

JavaScript files 115

of cascading style sheet files 115

of CSS files 115

- reordering *(continued)*
 - of JavaScript files 115
- request latency
 - minimizing 31
- request logging, and code elements 154
- request logging profile
 - and standard log formats 151
 - for NCSA Combined 152
 - for NCSA Common 152
 - for W3C Extended 152
 - overview 151
 - settings 153
- Request Logging profile
 - about 151
- requests
 - flow 16
 - management 15
- response codes
 - caching behavior 72
 - S codes defined 73
- responses
 - application of policy rules 67
 - assembly 68
 - classification 67
 - flow 16
 - management 16
- ROI reports
 - about byte savings 158
 - about Caching Bytes Saved reports 158
 - about Caching Requests Saved 158
 - about Client IBR'd Links Received reports 158
 - about Client IBR'd Links reports 158
 - about Compression Bytes Saved reports 158
 - about IBR savings 158
 - about ICC savings 159
 - about Image Optimization Bytes Saved reports 158
 - about Inlined Links reports 159
 - about Minification Bytes Saved reports 158
 - overview 157
- rules
 - meta tags 68
 - regular expressions 68

S

- sampling factor
 - optimizing 129
- S code
 - definitions 73
 - S10101 73
 - S10201 73
 - S10202 73
 - S10203 73
 - S10204 73
 - S10205 73
 - S10206 73
 - S10232 73
 - S10413 73
 - S11101 73
 - SO 73
- server connections
 - pooling of 33

- shaping policy 45
- SNMP
 - and bandwidth control policies 37
- source IP addresses
 - and OneConnect 33
- SPDY profile
 - overview 107
- SPDY protocol
 - purpose of 31
- static bandwidth control policies, *See* bandwidth control policies
- subnets
 - about discovery of 163
- symmetric data deduplication
 - described 161
- symmetric optimization
 - overview 17

T

- TCP connections
 - about optimization 12, 119
- TCP express
 - about optimizing mobile traffic 27
 - optimizing mobile traffic 26
- TCP profiles
 - about 26
 - and mobile traffic optimization 26–27
 - optimized for LANs 26
 - optimized for WANs 28
- throughput policy 41

U

- user credentials
 - and NTLM profiles 32

V

- variation rules
 - cache efficiency improvement 79
 - definition of rules parameters 80
 - management of conflicting rules parameters 81
 - overview 79
 - user-specific content 80
 - value groups 80
- video delivery optimization
 - about 131
 - about bit rate selection 131
 - about cache scoring 131
 - about caching popular content 131
 - about caching video segments 131
 - about global configuration 131
- video quality of experience
 - about 132
- video Quality of Experience
 - and mean opinion score 132
 - and MOS 132
- VIPRION
 - about acceleration in a cluster 97

W

- WAN traffic optimization
 - and TCP protocol [28](#)
- Web Acceleration profile
 - about [19](#)
 - settings [20](#)
- Web Acceleration Profile
 - tmsh statistics description [23](#)
- web application
 - optimization [13](#)
- Webp [128](#)

X

- X-WA-Info response headers
 - about [69](#)
 - about symmetric deployment [69](#)
 - A code [71](#)
 - N code [71](#)
 - P code [71](#)
 - RN code [71](#)
 - S code [71](#)
 - U code [71](#)
 - V code [70](#)