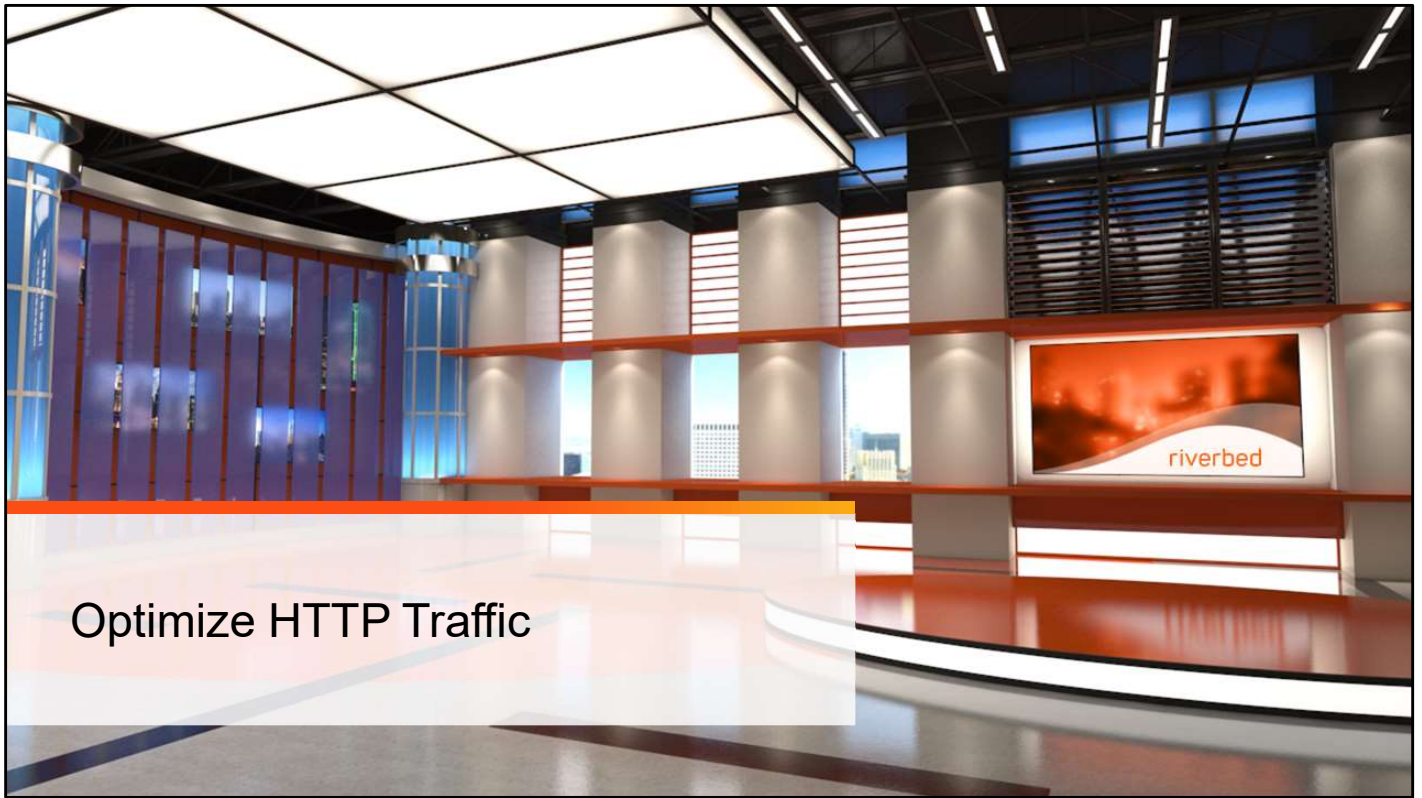


## **Part 3: OPTIMIZE Implement WAN Optimization**

- **Optimize HTTP Traffic**
- **Optimize SSL Traffic**
- **Customize Windows Traffic in an Active Directory Environment**
- **Optimize Other Application Layers**
- **Configure SteelHead SaaS Accelerator**
- **Describe SteelHead Cloud Solutions**
- **Monitor and Report on Solution Performance**

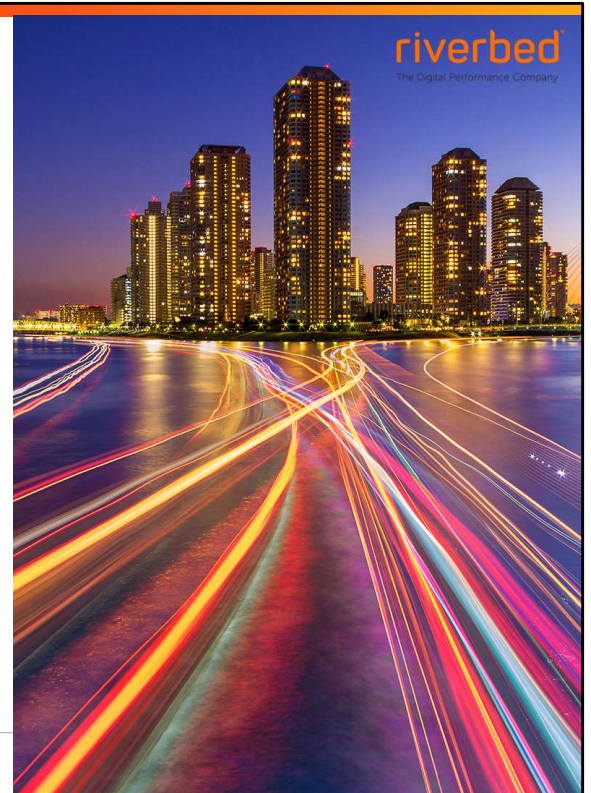


## Learning Objectives

After completing this module, you will be able to:

- Describe the HTTP protocol.
- Configure HTTP optimization.
- Describe HTTP automatic configuration.
- Describe HTTP-based Video Optimization.
- Describe the Web Proxy feature.
- Verify HTTP optimization.

© 2021 Riverbed Technology, Inc. All rights reserved.







## Key Points



HTTP is quickly becoming the protocol of choice for the majority of client-server applications.



Web pages are rendered line-by-line, which makes them sensitive to latency, just like any other application.



The SteelHead appliance has features that can accelerate HTTP, improving the performance of web applications, which are becoming key in distributed environments.

## The Growing Importance of HTTP for the Business

- Dedicated client applications are now becoming legacy.
- HTTP is now the application layer protocol of choice.
- Many global applications are now accessed via a simple web browser.
- This trend has accelerated since the introduction of HTML5.
- HTTP is used for much more than just SharePoint and the intranet.
- Performance of HTTP is now critical to modern business productivity!



© 2021 Riverbed Technology, Inc. All rights reserved.

riverbed 6

In the past, most applications worked with a standard client-server model. Take Exchange for example: The old, classic model was to have an Exchange Server and an Outlook Client communicating using MAPI/RPC.

The key point here is that there were dedicated Server/Client applications using specific protocols. You wanted to access your server application? Fine, download and install the appropriate client, which understands the application protocol.

However, as the internet has taken off during the last decade, more and more applications have moved away from this model and now just use HTTP.

Take Riverbed for example, the AppResponse appliance used to have a dedicated, Java-based, client application, which you needed to install. Now, if you want to perform some Web Transaction Analysis all you need is a browser.

HTTP is quickly becoming the application layer protocol of choice for most enterprise applications. A user in London, accessing an application server in San Francisco, will typically simply be using a browser such as Chrome, using HTTP.

This means that, in order for users in the remote location to be able to use that application, it is HTTP that needs optimizing to ensure that they can complete their transactions quickly and thus be as productive as possible.

## HTTP Methods

- A HTTP *method* indicates the action to be performed on a resource.
- Three commonly used methods are GET, POST and HEAD:
  - GET - Requests data from a specified resource on the server
  - POST - Submits data to be processed to a specified resource
  - HEAD - similar to a GET request, but the server responds only with the HTTP headers; the response body is empty



# HTTP Response Codes

The Server Responds to a Request Method with a Status Code

- Five classes of code:

- 1xx Informational
  - e.g., *100 Continue*
- 2xx Success
  - e.g., *200 OK*
- 3xx Redirection
  - e.g., *304 Not Modified*
- 4xx Client Error
  - e.g., *404 Not Found*
- 5xx Server Error
  - *500 Internal Server Error*



© 2021 Riverbed Technology, Inc. All rights reserved.

riverbed 8

In addition to the methods, the server will respond with a status code. There are five classes of status code:

## 1xx Informational

This class indicates a provisional response, consisting only of the Status-Line and optional headers, and is terminated by an empty line. e.g., *100 Continue*.

## 2xx Success

This class of status codes indicates the action requested by the client was received, understood, accepted, and processed successfully. e.g., *200 OK*.

## 3xx Redirection

This class of status code indicates the client must take additional action to complete the request. Many of these status codes are used in URL redirection. Two commonly used examples are:

- *301 Moved Permanently* - This and all future requests should be directed to the stated URL.
- *304 Not Modified* - Indicates that the resource has not been modified since the version specified by the request headers *If-Modified-Since* or *If-None-Match*. In such case, there is no need to retransmit the resource because the client still has a previously downloaded copy.

## 4xx Client Error

The 4xx class of status code is intended for situations in which the client seems to have erred. The most common of these is *404 Not Found*, indicating the requested resource could not be found.

## 5xx Server Error

This class indicates that the server failed to fulfill an apparently valid request. e.g., *500 Internal Server Error*.



## Cookies

### What is a Cookie?

- Small text file stored by a browser on the user's machine
- Plain text, not executables
- Set by the web server, the client sends it back with each subsequent request
- Used by the web server to track user sessions
  - Important for web applications and online shopping carts due to the stateless nature of HTTP
  - Very important for session persistence when using load balancers



HTTP is a stateless connection, by this we mean that, rather than having a single, established TCP connection for the duration of the user's session (like SMB), we have lots of short-lived TCP connections requested by the client browser, but from the server's perspective these will be totally independent.

The way in which session information is usually tracked is by using *cookies*. A cookie is a small file sent from the server and stored on the browser. When the browser makes further requests it sends back this cookie, allowing the web server to identify and track the session.

Cookies can either be:

**Persistent** – remain on the user's PC after the user closes the browser.

**Non-persistent** – discarded after the user closes the browser.

## How does a Web Page Load?

- HTML file is **rendered line-by-line**
  - <head> loaded with initial GET/fetch
    - CSS Stylesheets
    - JavaScript Libraries
  - <body> starts rendering in browser
    - CSS Layout
    - Headings
    - Paragraphs & Lists
    - Images
- CSS, JavaScript, Images fetched as called
  - When referenced inline, request initiated to server to fetch file
  - Multiple files? Separate connection for each
  - Rendering won't continue until file is fetched

The web page itself is stored in a HTML file, which references other objects (images, style sheets, scripts, etc.).

The browser renders this file line-by-line.

CSS, JavaScript, images are fetched as called.

When referenced inline, the request is initiated to the server to fetch the file.

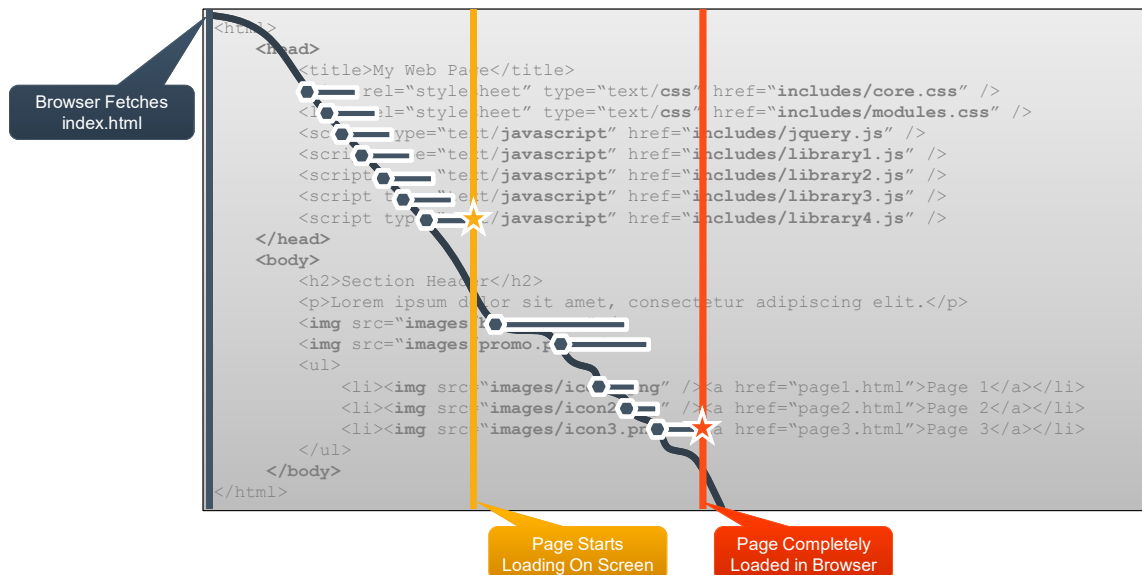
For multiple files, there is a separate connection for each.

Rendering won't continue until the file is fetched.

This gives us the classic Waterfall (see next slide).

## Waterfall: Rendering Line-by-Line

*This diagram is over simplified for educational purposes.*



© 2021 Riverbed Technology, Inc. All rights reserved.

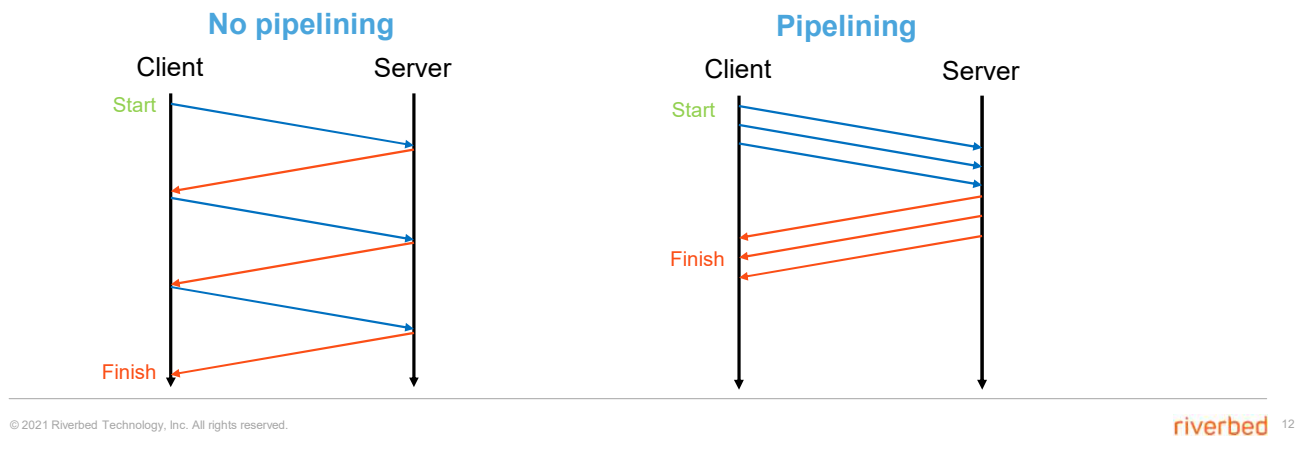
riverbed 11

Multiple TCP connections and pipelining are techniques which try and minimize this effect by using multiple TCP connections (obviously!) and sending requests in batches but not all browsers support pipelining and even those that do tend to have the feature disabled by default.

The consequences of this is that HTTP is just as latency bound as any other layer 7 protocol such as SMB.

## Multiple TCP Connections and Pipelining

- Requests sent in batches
- Can be used together with single or multiple HTTP connections



Most modern browsers establish two or more TCP connections to the server for parallel downloads. The concept is simple—as the browser parses the web page, it knows which objects it needs to download. Instead of sending the requests serially over a single connection, the requests are sent over parallel connections resulting in a faster download of the web page.

Another technique used by browsers to improve the performance is pipelining. Without pipelining, the client first sends a request to the server and waits for a reply before sending the next request. If there are two parallel connections, then a maximum of two requests are sent to the server concurrently. With pipelining, the browser sends the requests in batches instead of waiting for the server to respond to each individual object before sending the next request. Pipelining is used with a single HTTP connection or with multiple HTTP connections. Although most servers support multiple HTTP connections, some servers do not support pipelining. The diagram on the slide illustrates the effect of pipelining.

## HTTP Authentication

- Can be enabled to provide access control to the web page/site
- Many ways to perform authentication:
  - e.g., Certificates and smartcards
- Two most common mechanisms:
  - NTLM
  - Kerberos
- If the server supports both Kerberos and NTLM the browser tries Kerberos first then falls back to NTLM.



When the browser first connects to the server, it does not know whether the server has authentication enabled. If the server requires authentication, the server responds with a 401 Unauthorized message. Within the body of the message, the server indicates what kind of authentication scheme it supports in the WWW-Authenticate line. If the server supports more than one authentication scheme, then there are multiple WWW-Authenticate lines in the body of the message. For example, if a server supports both Kerberos and NTLM, the following appears in the message body:

**WWW-Authenticate: Negotiate**

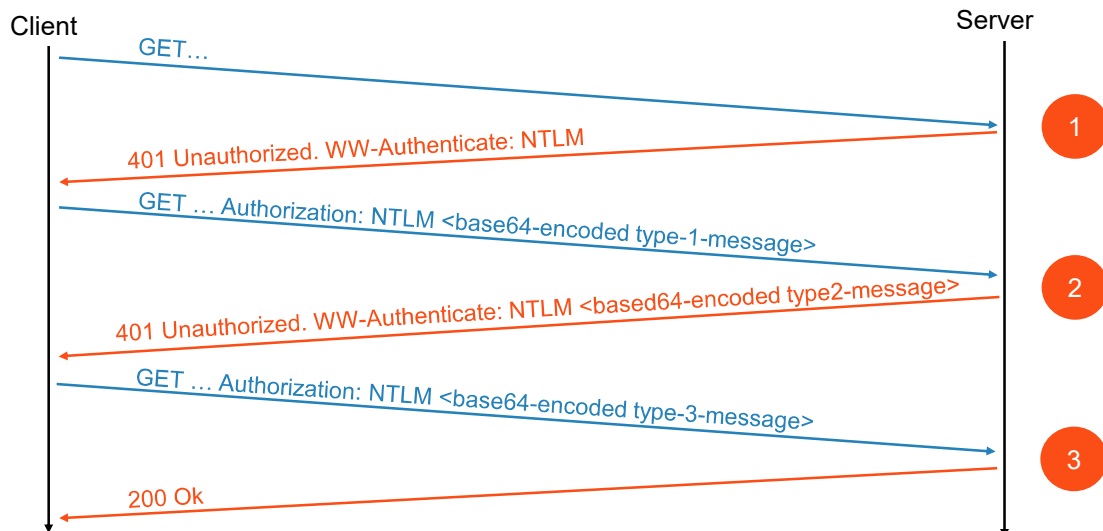
**WWW-Authenticate: NTLM**

When the browser receives the Negotiate keyword in the WWW-Authenticate line, it first tries Kerberos authentication. If Kerberos authentication fails, it falls back to NTLM authentication.



## Four-WAY NTLM Authentication

### Three Full Round Trips Required



© 2021 Riverbed Technology, Inc. All rights reserved.

riverbed 14

The following steps describe four-way NTLM authentication:

1. The client sends a GET request to the server.
2. The server, configured with NTLM authentication, sends back a 401 Unauthorized message to inform the client that it needs to authenticate. Embedded in the response is the authentication scheme supported by the server. This is indicated by the WWW-Authenticate line.
3. The client sends another GET request and attaches an NTLM Type-1 message to the request. The NTLM Type-1 message provides the set capability flags of the client (for example, encryption key size).
4. The server responds with another 401 Unauthorized message, but this time it includes an NTLM Type-2 message in the response. The NTLM Type-2 message contains the server's NTLM challenge.
5. The client computes the response to the challenge and once again attaches this to another GET request. Assuming the server accepts the response, the server delivers the object to the client.
6. Assuming a network on 200ms of round-trip latency, it would take at least 600ms before the browser begins to download the object.

## Authentication Types

### Per-Request and Per-Connection

- Per-Request Authentication
  - Client must authenticate every individual request
  - For 100 objects – client authenticates 100 times
- Per-Connection Authentication
  - If the client only opens a single connection then the client only needs to authenticate once
  - For 100 objects – client authenticates only once
- Default for IIS is:
  - Per-connection when using NTLM
  - Per-request when using Kerberos

© 2021 Riverbed Technology, Inc. All rights reserved.

riverbed 15

There is the concept of per-request and per-connection authentication with HTTP authentication. A server configured with per-request authentication requires the client to authenticate every single request before the server delivers the object to the client. If there are 100 objects (for example, .jpg images), it performs authentication 100 times with per-request authentication.

With per-connection authentication, if the client only opens a single connection to the server, then the client only needs to authenticate with the server once. No further authentication is necessary. Using the same example, only a single authentication is required for the 100 objects. Whether the web server does per-request or per-connection authentication varies depending on the software. For Microsoft's Internet Information Services (IIS) server, the default is per-connection authentication when using NTLM authentication; the default is per-request authentication when using Kerberos authentication.

Note that per-request authentication can add a significant overhead in high-latency links. If it takes 600ms per authentication request (200ms x 3), then for 100 objects this works out at a whole minute of overhead.

## Connection Jumping – An(other) IE issue

For those still using Internet Explorer (IE)...

- Specific to Internet Explorer when using parallel connections or pipelining with per-connection authentication
- For second or third connections, client tries to re-authenticate over second connection
- The re-authentication can be mistakenly initiated over the first, already authenticated connection, which can cause a reset
- Effectively turns per-connection authentication into per-request

© 2021 Riverbed Technology, Inc. All rights reserved.

riverbed 16

When the browser establishes the second, or subsequent, TCP connections for the parallel or pipelining downloads, it does not remember if the server requires authentication.

Therefore, the browser sends multiple GET requests over the second, or additional, TCP connections, without the authentication header. The server rejects the requests with the 401 Unauthorized messages.

When the browser receives the 401 Unauthorized message on the second connection, it is aware that the server requires authentication.

The browser initiates the authentication process.

Yet, instead of keeping the authentication requests for all the previously requested objects on the second connection, some of the requests jump over the first and are authenticated.

This is known as connection jumping.



## How Does Riverbed Optimize HTTP?

### ■ Key Optimization Features:

- Strip Compression
- Insert Cookie
- Insert Keep-Alive
- URL Learning
- Parse and Prefetch
- Object Prefetch Table

### ■ HTTP Authentication Optimization:

- Reuse Auth
- Force NTLM
- Strip Auth Header
- Gratuitous 401

#### Settings

##### Basic Tuning

- ☒ Strip Compression
- ☒ Insert Cookie
- ☒ Insert Keep-Alive

##### Caching

- ☒ Object Prefetch Table
- ☐ Stream Splitting

##### Prefetch Schemes

- ☒ URL Learning
- ☒ Parse and Prefetch

##### Authentication Tuning

- ☒ Reuse Auth
- ☒ Force NTLM
- ☒ Strip Auth Header
- ☒ Gratuitous 401

##### SharePoint

- ☐ FPSE
- ☐ WebDAV

☐ Enable Kerberos Authentication Support

☒ Enable Per-Host Auto Configuration

} seconds  
} seconds

The HTTP optimization blade was first introduced into RiOS way back in RiOS 4.1 with the feature URL Learning.

It has come on a long way since then and we will have a look at some of the additional optimization techniques that have been introduced in more recent versions.



## Key Optimization Features (1/2)

- **Strip Compression**
  - Removes the *Accept-Encoding* line from client request header
  - Response to the client is uncompressed, enabling optimal SDR/LZ
- **Insert Cookie**
  - Only needed if server does not support cookies
  - Inserts “rbt-http={random}” so server-SH can track different users
  - Cookie is removed by the client-SH
- **Insert Keep-Alive**
  - Need a permanent connection for prefetching
  - Inserts “connection: keep-alive” to client request
  - If server doesn’t support keep-alive, pre-fetching is not possible

Key optimization features:

**Strip Compression:** Removes the *Accept-Encoding* line from the request header before sending the request to the server. This ensures that the server sends the response to the client uncompressed, allowing SDR to work its magic.

**Insert Cookie:** The HTTP module relies on cookies to track different users, which enables it to perform URL Learning. If the server does not set a cookie then the SteelHead will insert one, which begins with *rbt-http=* followed by a random number. The SteelHead removes the cookie when it forwards the request to the server.

**Insert Keep-Alive:** This sends a keep-alive to the client response in order to keep a permanent connection which allows pre-fetching. The server must support keep-alive for this feature to work.

## Key Optimization Features (2/2)

### ■ URL Learning

- Saves client round trips for websites with static content and links
- Associates object references with the “Referer” header
- Builds a table relating base URLs to a number of object references
- Requests all referenced objects as soon as the base URL is accessed

### ■ Parse and Prefetch

- Saves round trips for websites with dynamic content and links
- SH parses the webpage looking for certain tags
- SH prefetches before client asks

### ■ Object Prefetch Table

- SH local data for cacheable content
- Looks for “if-modified-since” header

HTML Tags to Prefetch:

⊕ Add a Prefetch Tag   ⊖ Remove Selected

Tag Name	Tag Attribute
<input type="checkbox"/> base	href
<input type="checkbox"/> body	background
<input type="checkbox"/> img	src
<input type="checkbox"/> link	href
<input type="checkbox"/> script	src

© 2021 Riverbed Technology, Inc. All rights reserved.

riverbed 20

**URL Learning:** This is one of the core aspects of HTTP optimization. The SteelHead learns which objects are associated with a particular base URL, using the Referer field to build relationships between objects and the base URL page. When a SteelHead sees a request to a URL in its database it immediately sends requests for all objects associated with it, saving round trips for the browser.

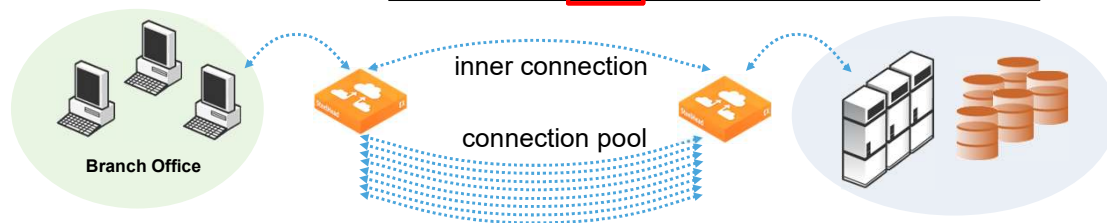
**Parse and Prefetch:** This allows the SteelHead to read a base HTML page and immediately request all of the objects associated with it before the client has received it and starts making its own request. This allows the objects to be served locally, saving trips across the WAN. This very much complements URL Learning, but is better for dynamic web content.

**Object Prefetch Table:** Although URL learning and parse and prefetch can request information from the HTTP server much sooner than the client would have requested the same information, the object prefetch table stores information to completely eliminate some requests on the WAN. The client browsers receive these objects much faster than they would if the object had to be fetched from the server. It also enables the client-side SteelHead to respond to If-Modified-Since requests from the client, again reducing WAN round trips.

## SteelHead Connection Pool

- Pre-established pool of inner connections
- Saves 1.5 round trips per connection
- HTTP benefits the most because these connections are typically short-lived

Filter: ((tcp.flags == 0x0012) && ! (tcp.options.rvbd.probe == 1))					
Protocol	Source	Source Port	Destination	Destination Port	Info
TCP	10.253.253.5	7800	10.18.38.236	49700	7800 > 49700 [SYN, ACK]
TCP	10.253.253.5	7800	10.18.38.236	49703	7800 > 49703 [SYN, ACK]
TCP	10.253.253.5	7800	10.18.38.236	51983	7800 > 51983 [SYN, ACK]
TCP	10.253.253.5	7800	10.18.38.236	52616	7800 > 52616 [SYN, ACK]
TCP	10.253.253.5	7800	10.18.38.236	54829	7800 > 54829 [SYN, ACK]
TCP	10.253.253.5	7800	10.18.38.236	57768	7800 > 57768 [SYN, ACK]
TCP	10.253.253.5	7800	10.18.38.236	57864	7800 > 57864 [SYN, ACK]
TCP	10.253.253.5	7800	10.18.38.236	57878	7800 > 57878 [SYN, ACK]
TCP	10.253.253.5	7800	10.18.38.236	57881	7800 > 57881 [SYN, ACK]
TCP	10.253.253.5	7800	10.18.38.236	57884	7800 > 57884 [SYN, ACK]
TCP	10.253.253.5	7800	10.18.38.236	57889	7800 > 57889 [SYN, ACK]
TCP	10.253.253.5	7800	10.18.38.236	57892	7800 > 57892 [SYN, ACK]
TCP	10.253.253.5	7800	10.18.38.236	57894	7800 > 57894 [SYN, ACK]
TCP	10.253.253.5	7800	10.18.38.236	57897	7800 > 57897 [SYN, ACK]
TCP	10.253.253.5	7800	10.18.38.236	57900	7800 > 57900 [SYN, ACK]
TCP	10.253.253.5	7800	10.18.38.236	57903	7800 > 57903 [SYN, ACK]
TCP	10.253.253.5	7800	10.18.38.236	57907	7800 > 57907 [SYN, ACK]
TCP	10.253.253.5	7800	10.18.38.236	57910	7800 > 57910 [SYN, ACK]
TCP	10.253.253.5	7800	10.18.38.236	57929	7800 > 57929 [SYN, ACK]
TCP	10.253.253.5	7800	10.18.38.236	60615	7800 > 60615 [SYN, ACK]
TCP	10.253.253.5	7800	10.18.38.236	60618	7800 > 60618 [SYN, ACK]
TCP	10.253.253.5	7800	10.18.38.236	60727	7800 > 60727 [SYN, ACK]



© 2021 Riverbed Technology, Inc. All rights reserved.

riverbed 21

Connection pooling pre-establishes 20 inner channel connections between each pair of SteelHeads. HTTP traffic benefits the most from connection pooling, although connection pooling is not specific to HTTP.

When the SteelHead requires an inner channel, it picks one from the pool and therefore eliminates the time for the TCP three-way handshake.

The reason HTTP traffic benefits the most is because those connections are typically short-lived.

## HTTP Authentication Optimization

### ■ Reuse Auth

- A prefetched object in the SH database can be delivered by an unauthenticated request.
- The session itself must have been authenticated!
- Without this option, the SH never delivers an object on an unauthenticated connection.

### ■ Force NTLM

- Removes “www-authenticate: Negotiate” from server 401 request
- Client then can *only* do NTLM.

### ■ Strip Auth Header

- Strips auth request header from an already authenticated connection
- Removes unnecessary round trips and solves *connection jumping* issues

### ■ Gratuitous 401

- Client SH caches 401 response and gives it to client on a GET without auth headers
- Removes one round trip

© 2021 Riverbed Technology, Inc. All rights reserved.

riverbed 22

Depending on the configuration, HTTP authentication can add a significant overhead, especially if each request is being authenticated. The following SteelHead features can accelerate this and can have a big impact when used across high-latency links:

**Reuse Auth:** This greatly speeds up the delivery of objects to the client when URL Learning and parse and prefetch are being used, by reusing the same credentials for an already authenticated session. Without this object, each connection would have to be authenticated between the client and server individually, losing any benefit of URL Learning or parse and prefetch.

**Force NTLM:** This forces the client to do NTLM authentication, which is much quicker as it is done on a per-connection basis, rather than Kerberos which uses per-request authentication.

**Strip Auth Header:** This option strips the auth headers for an already authenticated connection, which means that the objects can be delivered to the client without having to go through the whole authentication process again.

**Gratuitous 401:** The client-side SteelHead caches the server response. When the client sends the GET request without any authentication headers, it responds locally with a 401 unauthorized message and saves a round trip.

## Enable HTTP Kerberos Authentication

- To optimize HTTP connections using Kerberos authentication:
  1. Join the server-side SteelHead to the active directory domain.
  2. Configure a Replication User on the server-side SteelHead.
  3. On the HTTP page, select **Enable Kerberos Authentication Support**.
  4. Click **Apply**.

### Settings

☒ Enable HTTP Optimization

☐ Enable SteelFlow WTA

Object Prefetch Table Settings:

☒ Store All Allowable Objects

☐ Store Objects With The Following Extensions:

css.gif.jpg.js.png

☐ Disable The Object Prefetch Table

Minimum Object Prefetch Table Time: 60 seconds

Maximum Object Prefetch Table Time: 86400 seconds

Extensions to Prefetch: css.gif.jpg.js.png

☒ Enable Per-Host Auto Configuration

#### Basic Tuning

☒ Strip Compression

☒ Insert Cookie

☒ Insert Keep-Alive

#### Caching

☒ Object Prefetch Table

☐ Stream Splitting

#### Prefetch Schemes

☒ URL Learning

☒ Parse and Prefetch

#### Authentication Tuning

☒ Reuse Auth

☒ Force NTLM

☒ Strip Auth Header

☒ Gratuitous 401

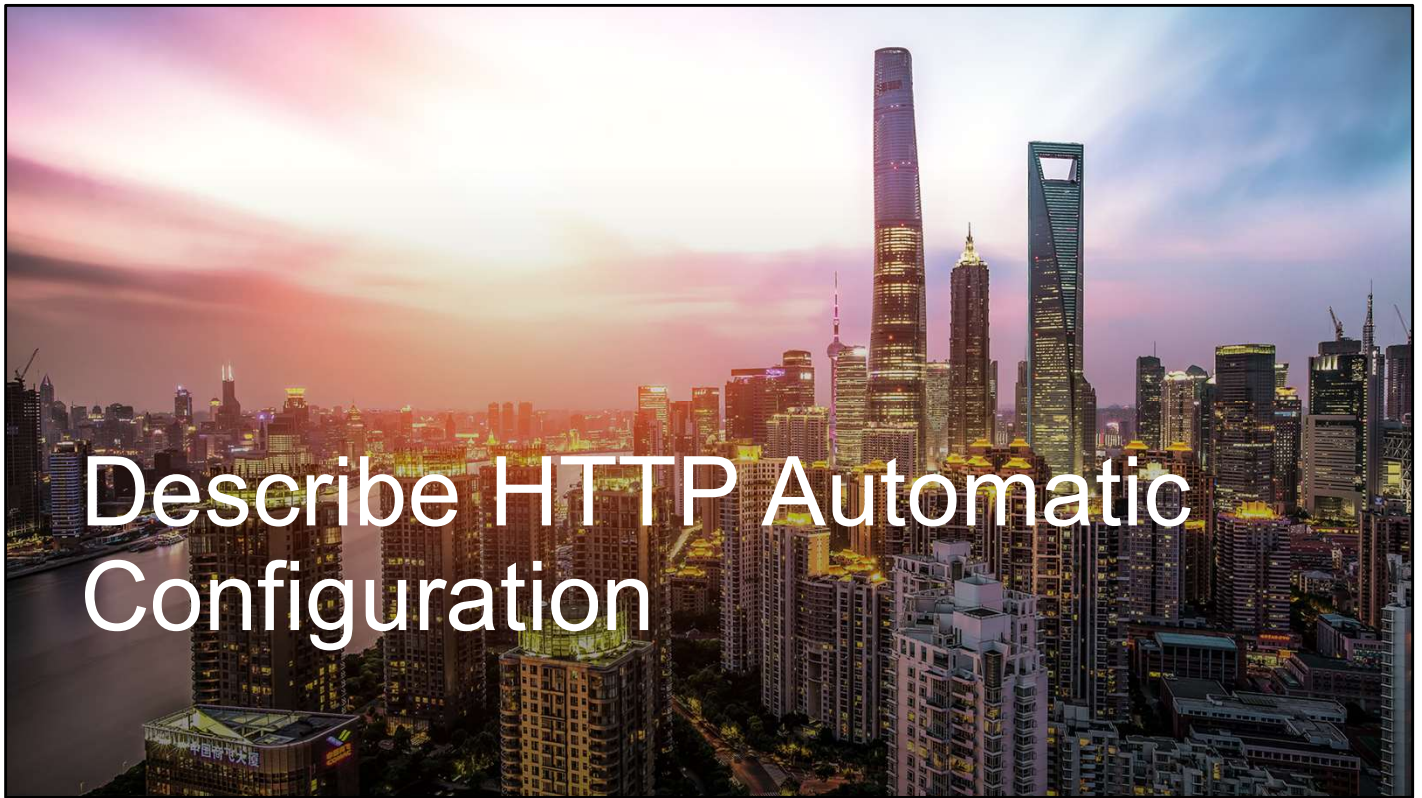
#### SharePoint

☐ FPSE

☐ WebDAV

☒ Enable Kerberos Authentication Support





## HTTP Automatic Configuration – Overview

- Automatically detects new HTTP web hosts and initiates an evaluation of the traffic statistics to determine the optimal settings.
- Rule sets are built automatically.
- The analysis is completed and configured independently on each client-side SteelHead.
- This feature is enabled by default.
- Four phases:
  1. Identification Phase
  2. Evaluation Phase
  3. Automatic Phase
  4. Static Phase

© 2021 Riverbed Technology, Inc. All rights reserved.

riverbed 25

HTTP Automatic Configuration can automatically detect new HTTP web hosts and initiate an evaluation of the traffic statistic to determine the most optimal setting. Rule sets are built automatically. The analysis of each HTTP application is completed and configured independently per client-side SteelHead. The feature is enabled by default and goes through four phases:

**Identification Phase:** Identify the HTTP applications

**Evaluation Phase:** Build up metadata, latency and throughput statistics. Just a subset of optimization features (e.g., insert keep-alive) are enabled during this time.

**Automatic Phase:** After the applications and statistics have been analyzed, the relevant features (e.g., URL Learning, parse and prefetch) can be applied as appropriate on a per-application basis.

**Static Phase:** Individual hosts or subnets can have custom settings applied, if required, overriding automatically learned settings.

## HTTP Automatic Configuration – Phases (1/2)

### ■ Identification Phase

- Applications are identified by hostname
  - Derived from HTTP request header
- Enables applications to be identified when multiple servers are used

### ■ Evaluation Phase

- Latency and throughput statistics built on HTTP applications
- Only a subset of features enabled during this stage
  - e.g., Object Prefetch Table, Insert Keep-Alive, Reuse Auth and Gratuitous 401

### ■ Automatic Phase

- Relevant features applied on a per application basis
  - URL Learning, Parse and Prefetch, Strip Compression and Insert Cookie

© 2021 Riverbed Technology, Inc. All rights reserved.

riverbed 26

### Identification Phase

HTTP applications are identified by a hostname derived from the HTTP request header. For example, host: sharepoint.example.com or www.riverbed.com. This makes applications more easily identifiable when an HTTP application is represented by multiple servers. For example, http://sharepoint.example.com is collectively resolved to multiple IP addresses, each belonging to multiple servers for the purpose of load balancing. A single entry encompasses the multiple servers.

### Evaluation Phase

The SteelHead reads the metadata and builds latency and throughput-related statistics on a per-HTTP application basis. After the SteelHead detects a predetermined number of transactions, it moves to the next phase. During the evaluation phase, the web server host has an object prefetch table, insert keep alive, reuse authentication, strip authentication and gratuitous 401 enabled by default to provide optimization. Strip authentication might be disabled after the evaluation stage if the SteelHead determines that the HTTP server requires authentication.

### Automatic Phase

The HTTP application profiling is complete and the HTTP application is configured. At this phase, prefetch (URL Learning and parse and prefetch), strip compression, and insert cookie optimization features are declared as viable configuration options for this application, in addition to the options from the earlier phase. Evaluation of prefetching is based on the time difference between the server-side SteelHead to the server and the server-side SteelHead to the client-side SteelHead. Prefetch is enabled if the time difference is significantly greater. Stripping the compression is enabled if the server-side SteelHead LAN bandwidth is significantly greater than the WAN of the client-side SteelHead. Insert cookie optimization is automatically enabled only when the server does not use cookies.

## HTTP Automatic Configuration – Phases (2/2)

- **Static phase**
  - Can insert custom settings for specific hosts or subnets

Server Subnet and Host Settings

Row Filters: ☒ Static ☒ Auto ☐ Auto (eval)

☒ Add a Subnet or Host ☐ Remove Selected

Server Subnet or Hostname:

<b>Basic Tuning</b>	<b>Authentication Tuning</b>
<input checked="" type="checkbox"/> Strip Compression	<input type="checkbox"/> Reuse Auth
<input type="checkbox"/> Insert Cookie	<input type="checkbox"/> Force NTLM
<input type="checkbox"/> Insert Keep-Alive	<input type="checkbox"/> Strip Auth Header
<b>Caching</b>	<input type="checkbox"/> Gratuitous 401
<input type="checkbox"/> Object Prefetch Table	<b>SharePoint</b>
<input type="checkbox"/> Stream Splitting	<input type="checkbox"/> FPSE
<b>Prefetch Schemes</b>	<input type="checkbox"/> WebDAV
<input type="checkbox"/> URL Learning	
<input type="checkbox"/> Parse and Prefetch	

© 2021 Riverbed Technology, Inc. All rights reserved.

riverbed 27

### Static Phase

You can insert custom settings for specific hosts or subnets. You can also select an automatically configured rule and override the settings. In RiOS 7.0 and later, you can insert hostnames (for example, [www.riverbed.com](http://www.riverbed.com)) along with specific IP hosts/subnets (for example, [10.1.1.1/32](http://10.1.1.1/32)). If you use a subnet instead of a hostname, you must specify the subnet mask.



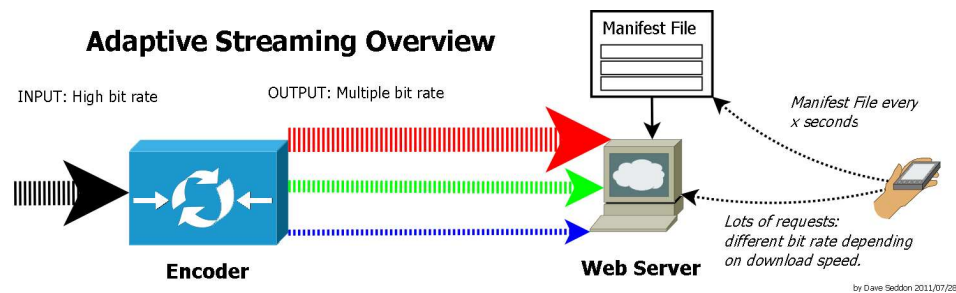


## Video-Specific Optimization Features

- In more recent versions of RiOS, there have been a number of features introduced which can greatly improve the performance of video traffic. These include:
  - Live streaming with split-streaming
  - On-demand with HTTP/Video prepopulation
  - On-demand video caching



# HTTP Stream Splitting – Streaming Video Overview



- Video optimization means HTTP *dynamic* streaming video protocols
  - Live Streams and Video On Demand
- SteelHeads understand these formats (v9.9+)
  - » Microsoft Smooth Streaming
  - » Adobe Dynamic Streaming for Flash
  - » Apple HTTP Adaptive Streaming
  - » MPEG-DASH
- Other Formats include
  - » Octoshape Multi-BitRate
  - » Digital Interactive Television via Internet
  - » QuavStreams Adaptive Streaming over HTTP
  - » upLynk

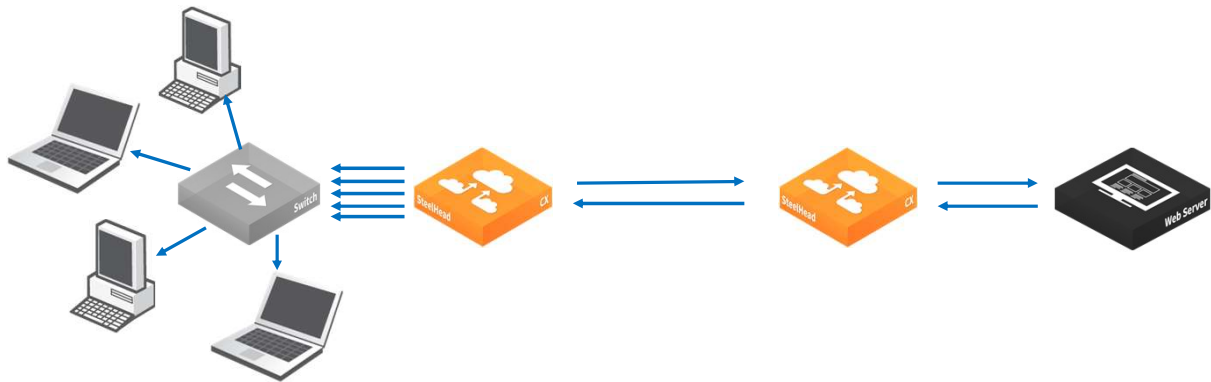
© 2021 Riverbed Technology, Inc. All rights reserved.

**riverbed** 30

This image is provided courtesy of Wikipedia.

## HTTP Stream Splitting – Stream Splitting & SteelHeads

### Stream Splitting Overview



- Stream splitting lets single stream of video serve many clients
- Designed for live streams
- Similar idea to multicast

© 2021 Riverbed Technology, Inc. All rights reserved.

riverbed 31

Stream splitting saves bandwidth: all clients maintain an active connection to the video source even though redundant streams are being removed from the WAN. Viewers with more bandwidth and processing power receive a higher quality video stream than viewers with less bandwidth and processing power.

## HTTP Stream Splitting – Recognized File Formats

### Live Stream Splitting Recognized File Formats

- Microsoft Silverlight: “Smooth Streaming”
  - Supports Live and on-demand video
  - Live streams matched in the URL on “.QualityLevels(”
  - Fragment is typically 2s of video
- Adobe HDS: “HTTP Dynamic Streaming”
  - Live streams matched in the URL on “/Seg?-Frag?”
- Apple HLS: “HTTP Live Streaming”
  - Cannot tell live stream from on-demand – stream split both
  - Matched in the URL on “Apple.\*CoreMedia”
  - Fragment is typically 10s of video
- MPEG-DASH: Dynamic Adaptive Streaming over HTTP



© 2021 Riverbed Technology, Inc. All rights reserved.

riverbed 32

RiOS uses HTTP stream splitting to optimize the following different live video technologies:

- Microsoft Silverlight (RiOS 7.0 or later)
- Adobe HTTP Dynamic Streaming (RiOS 7.0 or later)
- Apple HTTP Live Streaming (RiOS 8.5 or later)
- Dynamic Adaptive Streaming over HTTP (DASH), also known as MPEG-DASH (RiOS 9.8 or later)

## HTTP Stream Splitting – Considerations



- Stream splitting depends on all clients asking for the same video segment.
- What if a client asks for a lower bitrate?
  - New segment (different URL) and causes more requests
- What if a person hits “pause” for some reason?
  - Get out of sync due to cache timeout

## HTTP Stream Splitting – SteelHead Configuration

- Video optimization is part of HTTP
  - Single check box
  - Port 80 / 443 by default
  - In-path rules for more control
  - **Client side** feature only
- *Live Video Stream Splitting Report*, shows the data from the cache, server and video sessions

### HTTP Configuration Protocols > HTTP Configuration ⓘ

#### Settings

☒ Enable HTTP Optimization

☐ Enable SteelFlow WTA

Object Prefetch Table Settings:

☒ Store All Allowable Objects

☐ Store Objects With The Following Extensions:

☐ Disable The Object Prefetch Table

Minimum Object Prefetch Table Time:  seconds

Maximum Object Prefetch Table Time:  seconds

Extensions to Prefetch:

☒ Enable Per-Host Auto Configuration

<b>Basic Tuning</b>	<b>Authentication Tuning</b>
<input checked="" type="checkbox"/> Strip Compression	<input checked="" type="checkbox"/> Reuse Auth
<input checked="" type="checkbox"/> Insert Cookie	<input checked="" type="checkbox"/> Force NTLM
<input checked="" type="checkbox"/> Insert Keep-Alive	<input checked="" type="checkbox"/> Strip Auth Header
	<input checked="" type="checkbox"/> Gravitous 401
<b>Caching</b>	<b>SharePoint</b>
<input checked="" type="checkbox"/> Object Prefetch Table	<input type="checkbox"/> FPSE
<input type="checkbox"/> Stream Splitting	<input type="checkbox"/> WebDAV
<b>Prefetch Schemes</b>	
<input checked="" type="checkbox"/> URL Learning	
<input checked="" type="checkbox"/> Parse and Prefetch	

The basic idea is broadly similar to that of multicast. When you enable stream splitting, the first request for a video stream is sent out over the WAN, and the redundant requests are sent by the SteelHead when the first request is complete. As a result, only one copy of the stream is sent across the WAN no matter how many viewers are tuned in for the live stream. In RiOS 9.2 and later, the SteelHead identifies these streams using the video manifest file or video header metadata.

Enabling stream splitting is a single check-box under HTTP settings on the client-side SteelHead. There is also a *Live Video Stream Splitting* report, showing the data from the cache, server and video sessions.

## HTTP Stream Splitting – SteelHead Logs & CLI

- Look in the logs to see if its working

- When optimization service starts

```
http/inflightcache.NOTICE max cache size=20000000, delay=10000,  
inflight mode=0x1
```

- During optimization

```
[http/client.INFO] proc_get_req() <url> no request inflight...  
allowing to go  
[http/client.INFO] proc_get_req() <url> deferring inflight  
request
```

- Look on the CLI to see if it is working

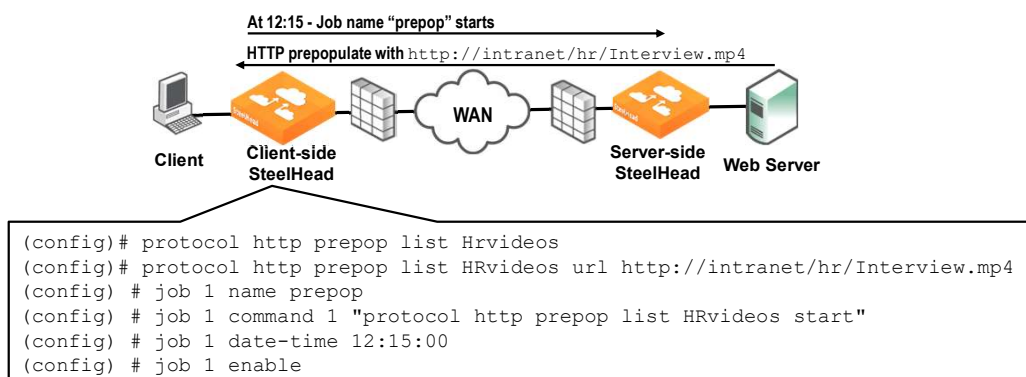
- **show protocol http stream-split stats**

```
Fragments fetched over WAN : 85  
Fragments served locally   : 683
```



## HTTP Prepopulation

- Pre-warms HTTP protocol on RiOS segstore
  - Delivers data content residing on Web server into segstore of SH relevant appliances
  - Provides remote users an enhanced viewing experience
- Use the job command on Client-side SteelHead to run the transfer during off hours



© 2021 Riverbed Technology, Inc. All rights reserved.

riverbed 36

HTTP prepopulation is an enhanced HTTP-based data delivery method. HTTP prepopulation delivers data to the remote site by using the HTTP protocol to pre-warm your RiOS segstore.

HTTP prepopulation uses the HTTP protocol to read and deliver data. The feature delivers any data content residing on a web server, including on-demand video streams at a remote site. Remote users can have an enhanced viewing experience. For more details on video optimization, see the *SteelHead Appliance Deployment Guide*.

HTTP prepopulation requires that you configure and connect the primary interface for full functionality.

HTTP prepopulation is only available through the command line. Configuration is only on the client-side SteelHead appliance.

To configure HTTP prepopulation, you create a list composed of URLs that contain the data you want optimized. You can configure up to 100 lists, and you can include an unlimited number of URLs within each list. For example, you can combine URL links to multiple Human Resource training videos in one list called HRvideos.



## Web Proxy – Feature Overview

- Single-ended feature, no server-side SteelHead is needed
- Client-side SteelHead must be able to access the Internet from the in-path interface
- The SCC is **required** for configuration
- Provides significant performance increase for HTTP traffic via localized serving
- Key use cases:
  - Microsoft update files
  - Videos delivered via YouTube

RiOS 9.1 and later includes the Web Proxy feature, which is the traditional single-ended internet HTTP-caching method of old, but enhanced by Riverbed for web browsing methodologies of today.

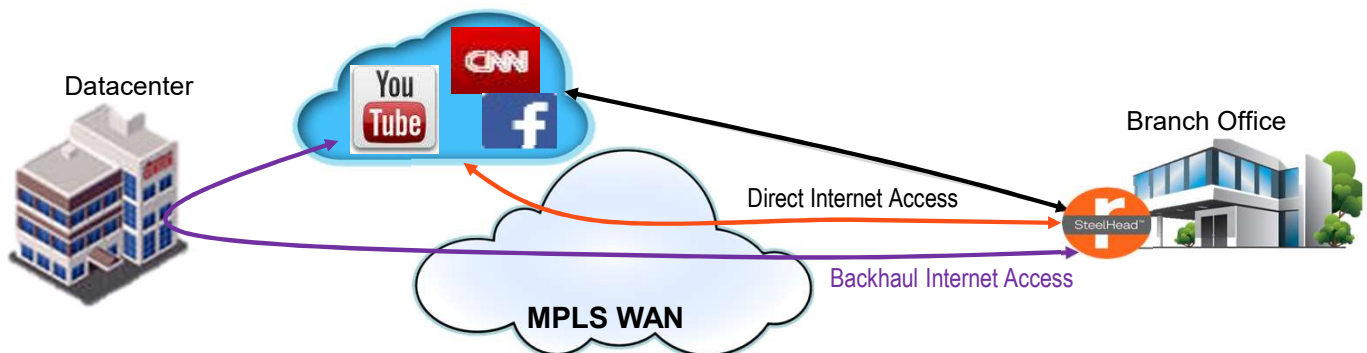
This feature enables SteelHeads to provide a local cache of web objects and files obviating the need for the requests to leave the branch.

Using the Web Proxy feature in the branch office provides a significant performance increase in accessing HTTP traffic due to the localized serving of this traffic and content from the cache.

This is particularly useful for cases where multiple users are downloading large bulky Microsoft update files or watching technical videos on YouTube.

Furthermore, multiple users accessing the same resources receive content at LAN speeds while freeing up valuable bandwidth.

## Web Proxy – Target Deployment Models



- Case #1: Internet Connections backhauled through corporate datacenter
- Case #2: Direct to Internet connections
  - Internet breakout at the branch
  - Internet breakout from within the managed SP network

## Web Proxy – Implementation Requirements

- **SCC**
  - Required to centrally configure and manage
  - Can centrally view and monitor the cache-hit data collected across sites
- **SteelHead**
  - Hosts the Web Proxy configuration and provides the proxy and cache services for each location
  - Supports physical or virtual in-path
  - Web Proxy is not supported on the xx50 or xx60 models
  - Web Proxy is only supported on the SteelHead virtual models VCX-SUB-10 to VCX-SUB-90
- **DNS**
  - DNS resolution via the Primary interface is essential for successful operation

© 2021 Riverbed Technology, Inc. All rights reserved.

riverbed 40

Web Proxy is critically dependent on DNS resolution, specifically Reverse DNS lookups sourced from the Primary interface, for appropriate HTTP/HTTPS proxy services to occur.

Because the SteelHead must successfully resolve hostnames to be cached and proxied the Primary interface of the SteelHead must be configured with valid IP address and DNS information.

In addition, the interface must be in an active state (even when it is not used by your supported deployment model). Make sure that the SteelHead DNS configuration and the Primary interface on the SteelHead are both configured and active.

### IP Address Support

IPv4 only

Non-RCPE 1918 (Public) addressing supported by default

RFC 1918 (Private) address can be used via additional in-path rule configuration

Customer's intranet and internal addressing

TCP Port Support

TCP 80 (HTTP) and 443 (HTTPS) default

Non-standard ports can be supported via additional in-path rule configuration

## Web Proxy – Interaction with SaaS Traffic

- Web Proxy and SaaS Accelerator
  - SaaS does not use Web Proxy
    - SaaS uses traditional dual-ended interception (DEI) HTTP optimization
- Web Proxy can co-exist with SaaS Accelerator
  - SaaS traffic to services such as Office365 and Salesforce.com bypass the Web Proxy feature

## Web Proxy – Internet Video Caching Services

### Ergo: YouTube

- Certain video services, specifically YouTube, can be cached using the Web Proxy.
  - Static video content that is cache-eligible
  - Think video *files* not *streams*
    - YouTube is not really streaming video to your browser
- Most Internet browsers are supported when accessing YouTube.
  - Firefox will not leverage any video cache features because it makes some header manipulations that the Web Proxy feature cannot identify.
- YouTube video typically relies on HTTPS, so it requires global whitelist entries for both [\\*.youtube.com](https://*.youtube.com) and [\\*.googlevideo.com](https://*.googlevideo.com).



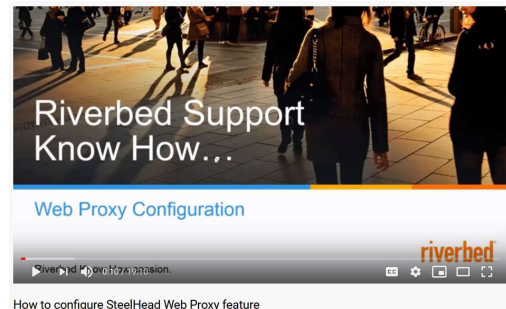
## Web Proxy – Additional Information

- How to configure SteelHead Web Proxy feature

- <https://www.youtube.com/watch?v=kTQBtpbrUmM>

- SteelHead documentation

- SCC Deployment Guide
  - SCC User Guide
  - SteelHead User Guide

















## Verify HTTP Optimization

### The Current Connections Report

- Check the Current Connections Report.
- Optimized HTTP connections display **HTTP** in Application column.

CT	Notes	Source:Port	Destination:Port	LAN kB	WAN kB	Reduction	Start Time	Application
»*	  	10.1.21.110:49530	10.1.31.130:23161	3,575	2,145	40% 	2019/05/24 08:01:00	FTP-DATA
»*	  	10.1.21.110:49421	10.1.31.130:445	27,283	10,126	62% 	2019/05/24 07:54:14	CIFS
»*	  	10.1.21.110:49533	10.1.31.130:80	6,120	2,640	56% 	2019/05/24 08:01:19	HTTP

- *Note: SteelHeads automatically recognize HTTP traffic on port 80 and 8080. If using a different port, an in-path rule will be needed.*

## Enable HTTP Optimization for Custom Applications

- Select the port in the *Destination* section.
- Specify a *Latency Optimization Policy* of **HTTP**.

▼ Add a New In-Path Rule   ◀ Remove Selected Rules   ⬆ Move Selected Rules...

---

Type: Auto Discover ▼

Enable Email Notification: ☐

Source: { Subnet: All IPv4 ▼

Destination: { Subnet: All IPv4 ▼  
Port: Specific Port ▼  (1 - 65535)  
Domain Label: n/a ▼

VLAN Tag ID: all

Preoptimization Policy: None ▼

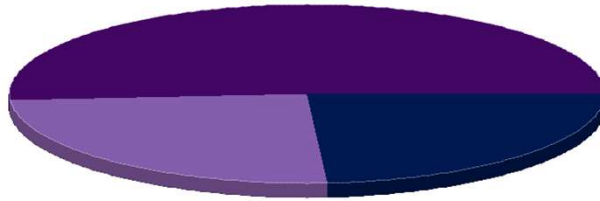
Latency Optimization Policy: HTTP ▼

Data Reduction Policy: Normal ▼

# Determine HTTP Bandwidth Reduction

## Traffic Summary Report

Traffic Summary [Networking > Traffic Summary](#) [?](#)



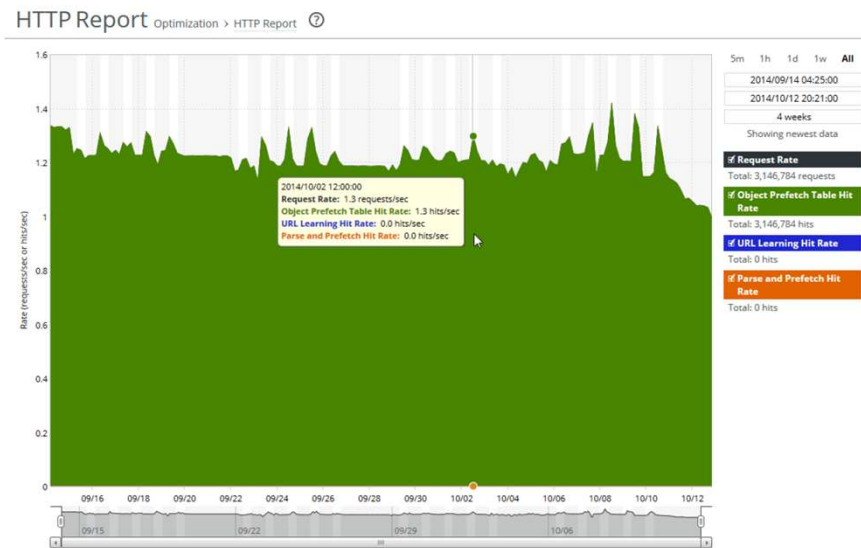
Period:  Type:  Traffic:  Refresh:

Port	Reduction	LAN Data	WAN Data	Traffic %
Total Optimized Traffic	<div><div></div></div> (43.29%)	113.8 MB	64.5 MB	--
80 (HTTP)	<div><div></div></div> (39.41%)	58.5 MB	35.4 MB	51.38%
21 (FTP)	<div><div></div></div> (32.99%)	28.7 MB	19.2 MB	25.21%
445 (CIFS:TCP)	<div><div></div></div> (62.89%)	26.6 MB	10125.3 kB	23.41%

# View Hit Rate for HTTP Optimization

## The HTTP Report

- Displays the following statistics:
  - Request Rate
  - Object Prefetch Table Hit Rate
  - URL Learning Rate
  - Parse and Prefetch Hit Rate



© 2021 Riverbed Technology, Inc. All rights reserved.

riverbed 48

## Putting it all Together – Real World Use Case

Scenario	An insurance company based in the UK relocated their primary call center from Newcastle in the UK to India. Operators based in India accessed an application server in the UK over HTTP using a web browser.
Issue	Due to both the bandwidth and latency from the UK to India (200-300ms) the application was almost unusable, causing many problems for the company and a major loss of revenue due to a reduced number of transactions per hour. This caused such a problem that they were reconsidering their decision to relocate.
Solution	After a successful Proof of Concept, the customer purchased the Riverbed solution, which enabled the application to run as productively in India as it did in the UK, with an ROI of only a few months.



## HTTP Optimizations

In the available HOL & lesson:

- HOL1158: Optimize Videos with Stream Splitting & HTTP Prepopulation
- Optional eLearning Activity: Optimize Web Traffic Using Web Proxy; lesson on details of Web Proxy concept & configuration

**Duration: 30 - 60 minutes**

- HOL1158
- Optional: Web Proxy detail lesson



*eLab system: link and access details provided in your course confirmation email*

## Module Review

You should now be able to:

- Describe the HTTP protocol.
- Configure HTTP optimization.
- Describe HTTP automatic configuration.
- Describe HTTP-based Video Optimization.
- Describe the Web Proxy feature.
- Verify HTTP optimization.

© 2021 Riverbed Technology, Inc. All rights reserved.

**riverbed**  
The Digital Performance Company