HTTP compliance verification

White paper

Contact:

Networking Products Division
HCL Technologies Ltd.
49-50 Nelson Manikkam Road
Chennai- 600 029, INDIA

May 2002

http://cdn.hcltech.com

Contents

Introduction

Along with the astounding increase in Internet traffic, several problems were discovered in the de-facto standard of the web traffic, HTTP/1.0 protocol. To solve these, a latest version of the Hypertext Transfer Protocol HTTP/1.1 was introduced in June 1999. (RFC 2616 by IETF HTTP Working Group). With the release of the new version, the components that make the Internet were expected to be compliant with the protocol.

Any application, whether it is a server, proxy or a user agent, is said to fully HTTP/1.1 compliant if it complies with all MUST, SHOULD and MAY conditions of RFC2616. If the application complies with all the MUST conditions but not all the SHOULD and MAY conditions then it is said be conditionally compliant. Main advantage of ensuring RFC compliance of an application is that it enables interoperability with products from multiple vendors.

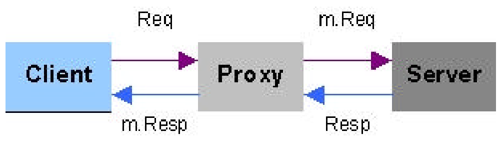
In the context of content delivery networks (CDNs), servers, proxies and caches need to be verified for HTTP compliance. This document details the context and methodology of the verification process.

HTTP compliance of the edge devices

The edge devices in a CDN include caches and proxies. To verify whether a proxy is fully HTTP/1.1 compliant, HTTP requests are simulated taking each of RFC condition into account. The behavior of proxy using logs and requests/response it sends to server/client are analyzed. For example:

- Send a HTTP request with the version higher/lower than that the proxy supports and verify whether it downgrades/upgrades the requests.
- Check whether the proxy truncates the leading and trailing zeros in HTTP version string in the HTTP request.

Similarly, to verify compliance of proxy with respect to the response it handles, various responses from the web server are simulated and its behavior is analyzed.

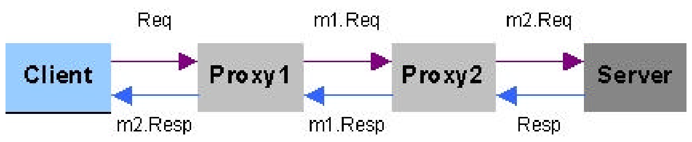


HTTP Compliance verification with single proxy

For example:

- Send a request with q value 0 for a particular content type, simulate server response in the non-acceptable content type and check whether the proxy sends the appropriate response.
- Send a looping request to the proxy and check whether it can recognize the request and sends the appropriate response to the client.

In the above diagram, Req and m.Req are compared to make sure that the proxy modifies the request correctly before it is forwarded to server/another proxy. Resp and m.Resp are also compared to ensure that proxy sends the correct response to end client. Compliance in a topology that has a hierarchy of caches/proxies can also be verified.

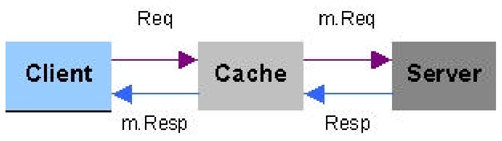


HTTP Compliance verification with hierarchy of proxies

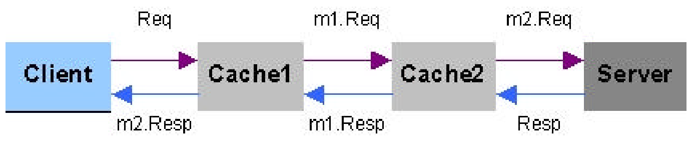
Verifying whether a cache is HTTP/1.1 compliant, includes:

- Checking the ability of cache to distinguish between cacheable and non-cacheable content. This is done by requesting static html pages and dynamically generated pages.
- Checking whether cache gives valid or stale data. This is verified by having various cache control headers in request and response.
- Checking its ability to request/serve partial content. This is verified by sending byte range request and its response.

HTTP/1.1 compliance could be verified in a setup that has either a single cache or hierarchy of caches.



HTTP Compliance verification with single cache



HTTP compliance verification with hierarchy of cache

For example:
Client has sent byte range request. But server has sent entire response. In this scenario, whether cache can send only requested bytes to client instead of sending the entire response is verified.

Steps to verify/implement HTTP compliance

- Identify the RFC conditions that are applicable to the application (depending on whether it is a proxy, cache or server)
- Develop positive and negative test cases for each condition.
- Develop CGI scripts to simulate various server responses.

The test cases can be tested either manually or using scripts.

Levels of tests

To confirm HTTP compliance, there can be three levels of tests:
Category 1: Checking for conformity to all MUST conditions
Category 2: Checking for conformity to all SHOULD conditions
Category 3: Checking for conformity to all MAY conditions

Category 1 compliance

HTTP/1.1 specification clearly indicates that all servers must implement GET and HEAD methods. In the first category of the test, GET and HEAD methods with required modifiers are checked. It also checks for the absence of required Host header. Any compliant HTTP/1.1 server has to succeed in these tests. It should be noted that the version number in an HTTP message is a hop-by-hop header (as opposed to an end-to-end header). If tests are directly from client to origin server, exact version number of the origin server will be obtained. Most of all HTTP requests made to Web servers are GET, the basic way to request a resource on the Web. Use of modifiers with GET (such as If-Unmodified-Since), however, are new to HTTP/1.1. These tests are to verify that servers respond with (new response code) *412 Precondition Failed*, when the precondition fails.

Category 2 compliance: Selective implementation of features

In the second category tests, compliance to improvement features added to HTTP/1.1 is verified. The protocol permits servers to selectively implement these features, but with the general expectation that a HTTP/1.1 server is compliant to these features. Major aspects of these features that are tested are persistent connection handling, pipelining of requests and range requests.

One of the major innovations in HTTP/1.1 was the introduction of persistent connections. In the older version, connections were established for a single request/response exchange. This necessitated TCP setup and teardown for each request, resulting in perceived user latency. This also caused traffic of additional packets in the network. The situation turns for worse in case of accessing a page with numerous images in it, as it required multiple TCP setups and teardowns. Most HTTP transactions are short and TCP handshakes took up most of the overall time. Persistent connections are default in HTTP/1.1, though servers or clients could close the connection after the first exchange. In fact, downloading all the embedded images in a single persistent connection has the best performance.

Another improvement is the ability to pipeline a stream of requests without waiting for any response from the server, eliminating the round trip time of waiting for acknowledgments of previous requests. However, the server sends responses in the order of requests received. At the same time, persistent connections without pipelining can in some cases adversely affect the performance. Also, in some cases multiple parallel non-persistent connections are found to be better. However, this advantage came at a cost, which though minimal to the browser, is high for the server because it has to deal with multiple simultaneous connections from each client. Persistent connections with pipelining provided the best combination to reduce latency and overall number of packets. One of improvements that was introduced in HTTP 1.1, the ability to hold the connection open beyond a single connection and to handle pipelined requests is one thing that can be verified.

Another important improvement in HTTP/1.1 is the ability to request byte ranges of resources rather than the full contents. This enables requiring just the tail of a growing resource, prefetching headers of resources of content types like images to begin outlining before actually fetching the content. Range requests also facilitate recovering from aborted connections and transfers. If parts of the resources are cached, only the missing parts need to be obtained.

Category 3 Compliance

The tests in this category are intended to verify if some of the features that are incorporated in the HTTP/1.1 are implemented in the server. Some of them are the OPTIONS method, the Expect/Continue mechanism and conditional requests.

OPTIONS method

The OPTIONS method indicates the capabilities of origin server. If a resource is specified, optional features applicable to that resource alone is to be returned. The TRACE method purely runs a loop back test of the message included in the request and is simply a way to see if the server received exactly what was sent from the client. The server is supposed to return the request it received in the response body.

Expect/Continue mechanism

This mechanism was implemented to prevent clients from sending large bodies in PUT/POST requests that might not be accepted by a server. Through this mechanism, clients can check with the server beforehand whether the request is supported. A client would send just the header (without a body but with a content length indicator) including a request header *Expect: 100-Continue*. If server accepts the request, it would reply with a *100 Continue* status response. On receiving the response from the server, client can send the body. If server does not accept the request, it will send a *401 Unauthorized* or a *417 Expectation Failed* response.

Conditional requests

HTTP/1.1 introduced several new conditionals to improve the caching model. Instead of simple *Last-Modified* timestamp check that HTTP/1.0 provided in *GET If-Modified-Since* request, presence of opaque strings in the form of entity tags permits a more general model. If several instances of a resource are maintained at the server and cached at a proxy, the proxy could check if any of its cached instances are current by including conditional headers such as *If-Match*. Additionally, an *If-Unmodified-Since* conditional permits a resource to be sent only if it has not changed since the indicated date.

Also, HTTP applications have permitted three date formats. While HTTP/1.1 clients and servers have to accept all three formats for compatibility with HTTP/1.0, they can only generate the RFC 1123 format for representing date values in header fields.

Test methodology

With the focus on HTTP/1.1 compliance of edge devices, RFC conditions that are relevant to cache and proxy are also considered. HTTP compliance is tested by writing scripts to send customized requests to edge device either directly or through a proxy and analyzing how edge-devices interpret and modify hop-to-hop headers. In addition, testing involved checking whether cache could send the expected response to client.

Test environment

The following environment was used to test the HTTP compliance:

- Web Servers running on Linux platform.
- Client scripts written in perl
- CGI scripts written in perl to simulate various server responses.
- An automated test environment in which either a selected set of tests or the entire test suite can be run.

Hardware Requirements

- 256 MB RAM
- 1 GHz Pentium III Processor
- 20 GB Hard disk

Software requirements

- Linux 6.2 (2.2.14-5) or later
- Perl 5.005 or later
- Apache 1.3.20 or later
- Netscape 4.1 or later

Conclusion

With the release of HTTP/1.1 as the web protocol, servers, proxies and caches need to be made compliant. It is all the more important in the case of Content Delivery Networks, as they form the backbone for efficient content delivery to web users. Verification of compliance to HTTP protocol of these CDN components was done with a defined framework of testing.

About HCL Technologies

HCL Technologies, with a revenue of US$ 297 millions, is one of India's leading IT services companies, providing a broad range of services to clients worldwide. Services include Technology Development, Software Product Engineering, Networking & Application Services and Business Process Outsourcing.

HCL Tech focuses on technology as well as research & development outsourcing, with the objective of working with clients in areas at the core of their business. The focus on such mission critical projects and the ability to provide services throughout the life cycle of client products, from conceptualization to ongoing development and maintenance, enables HCL Tech to build long-term relationships with customers. These include software and hardware companies as well as large and medium sized organizations, across diverse industries around the world. Market leaders like Cisco Systems, Novell, RSA Security, KLA Tencor etc. feature in the reputed list of clients of HCL Tech.

HCL Tech delivers services through an extensive offshore software development infrastructure in India and a vast global marketing and project network that enables scalable, flexible and cost-effective delivery. The company's offshore model involves delivery of outsourcing services to clients abroad, by technical professionals located at the software development centers in India and may also include onsite work at the client site, on a short-term project-by-project basis. As of March 31, 2002, HCL Tech had 5945 employees including JVs and subsidiaries. The company is thus able to capitalize on the advantages inherent to the Indian IT sector, including access to a large pool of skilled Indian technical professionals who deliver high-quality, globally competitive services at a significantly lower cost than in the United States.

The offshore model fosters strong client relationships because some clients also make substantial capital investments in the dedicated offshore development centers set up exclusively for them. HCL Tech's extensive marketing network comprises 21 marketing offices in 14 countries. Since inception, HCL Tech has emphasized the importance of building skills in emerging technologies by focusing on research and development activities for clients. The company's R&D heritage stems partly from the early efforts of several key senior personnel who were actively involved in research and development related to the design of computer hardware and systems software products for the Indian market in the 1980s. HCL Tech continues to develop its IT services business by leveraging on the unique skills and know-how of these executives and other employees.