Network Working Group
Internet-Draft
Expires: January 5, 2001

M. Nottingham
Akamai Technologies

July 7, 2000

Requirements for Demand-Driven Surrogate Origin Servers
draft-nottingham-surrogates-00

Status of this Memo

Copyright Notice

Abstract

This document states requirements for demand-driven surrogate origin servers, also known as reverse proxies and Web accelerators.

1. Introduction

A surrogate origin server (also known as a reverse proxy or HTTP accelerator) is a device that authoritatively serves requests on behalf of an origin server (known as its master origin server)[1].

Demand-driven surrogate origin servers are populated by the traffic flowing through them; when a client requests an object which is not resident, they will fetch it from the master origin server.

It may be useful to conceptualize a demand-driven surrogate as an origin server that happens to be populated via the HTTP on the back end.

In many ways, they are similar to proxy/caches, and often leverage proxy/cache software. However, surrogates serve content authoritatively, and therefore take the role of an origin server, not a proxy, to downstream clients.

Unfortunately, the use of a proxy/cache as a surrogate origin server introduces several problems in protocol implementation, due to this changing of roles. This document attempts to rectify such inconsistencies.

Additionally, master origin server administrators usually have a greater degree of control over the activity and use of surrogates than they would over proxies. Because of this close relationship, more precise control over the behavior of the surrogate can be given to the administrator.

This document specifies acceptable mechanisms for doing so.

1.1 Requirements

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in RFC 2119[4].

An implementation is not compliant if it fails to satisfy one or more of the MUST or REQUIRED level requirements. An implementation that satisfies all the MUST or REQUIRED level and all the SHOULD

level requirements is said to be "unconditionally compliant"; one that satisfies all the MUST level requirements but not all the SHOULD level requirements is said to be "conditionally compliant".

1.2 Terminology

This document uses terms defined and explained in the WREC Taxonomy[1], and the HTTP/1.1 specification[2]. The reader is

expected to be familiar with both.

In this document, the term "surrogate" is shorthand for a demand-driven surrogate origin server, unless explicitly stated otherwise. Similarly, "origin server" refers to a surrogate's master origin server.

2. Overview of Demand-Driven Surrogate Origin Servers

2.1 Uses and Characteristics

In normal operation, demand-driven surrogate origin servers are deployed and maintained by (or on behalf of) the publisher of a Web site, rather than directly for end users (as a proxy would be). This is often done for a number of reasons, including (a non-exhaustive list):

- Reduction of load on the master origin server
- Reduction of network traffic to the master origin server
- Distribution of objects, in order to improve perceived latency by storing them closer to end users
- Introduction of content transformation or other value-added services

Surrogate deployments may vary in several ways, including:

- Proximity - surrogates may be deployed close to the master origin server to reduce load on it, or near end users to reduce network traffic and improve perceived latency.
- Selection of surrogate objects - entire Web sites may be routed through surrogates, or a subset of a site's objects may be nominated for publication through them, depending on the effect desired, and the nature of the surrogate.
- Number of surrogates - surrogates may be deployed in any number. Localized surrogates may use any of a number of mechanisms to distribute requests between them, while distributed surrogates

usually use wide-area DNS load balancing.

By their nature, surrogates are never the parent or child of other surrogates. However, they MAY have such relationships with proxy/caches.

2.2 General Operation

2.2.1 Configuration

In order to accept and properly handle requests on behalf of a master origin server, a surrogate needs to be aware of its master's identity, and the profile of traffic that will be served on its behalf.

Additionally, it may be desirable to configure surrogates with other information, including:

- Any encryption or authentication information required by the master origin server
- Default object handling information, including coherence
- Specific object handling information
- Other special instructions to the surrogate

Surrogates may be configured by a variety of mechanisms, including manual, out-of-band, or vendor-specific.

Some types of surrogate configuration may be communicated in-band, by HTTP headers described in this document. However, such information is not neccessarily limited to that form of communication.

Manual and out-of-band configuration mechanisms may vary in implementation; specification of them is out of scope for this document.

2.2.2 Request Handling

A surrogate is configured to forward traffic to a master origin server, so that the hostname of the surrogate may be used in published URLs.

A surrogate MAY be configured to forward traffic to multiple master

origin servers by using the Host request header to differentiate requests. In this scenario, requests without a Host header SHOULD be replied to with a 502 Gateway Error response status code.

Surrogates MUST accept Absolute-URI[3] as well as Relative-URI requests and forward them to the master origin server, as configured. They MUST NOT forward Absolute-URI requests to origin servers that they have not been configured to serve.

Surrogates MAY use encryption (SSL or TLS) on downstream, upstream or both connections.

2.3 Origin Server to Surrogate Optimizations

Surrogates serve content on behalf of nominated origin servers, implying that the origin server administrator has access to configure, monitor and receive logs from the surrogate.

Because of this, a greater degree of trust exists between them than there would be between an origin server and third-party proxies. This allows modification or extension of the relationship between them, to offer greater control and functionality.

2.3.1 Separation of Coherence

Origin server administrators are wary of trusting third-party caches to keep objects coherent, because they do not always implement coherence in a predictable or correct manner.

Surrogate coherence behavior can be both predicted and tested by origin server administrators. However, there is still need to be able to describe object coherence to downstream caches.

This leads to the need for separate coherence mechanisms; one between the master origin server and surrogates, and another between surrogates and their clients.

This is accomplished by defining new, surrogate-specific mechanisms, while traditional coherence mechanisms retain their meaning for downstream caches. While the new mechanisms are introduced as HTTP headers here, they MAY also be communicated by separate configuration of the surrogate.

2.3.2 Protocol Feature Manipulation

Surrogates MAY add end-to-end protocol features that are not supported by the origin server, in order to offer greater functionality to downstream clients. For example, a surrogate could add ETag validators to objects, to improve downstream cacheability.

Surrogates may also implement hop-by-hop mechanisms (such as transfer encoding for compression and persistent connections) that are lacking on the master origin server, to offer improved quality of service to their clients.

When offering extended end-to-end features, surrogates MUST defer to support on the origin server; if a feature is present there, it cannot be overridden by the surrogate implementation.

2.4 Problems Introduced by Use of Proxies as Origin Servers

2.4.1 Dates and Age Calculation

In HTTP/1.1[2] The Date response header is required to reflect the time that an object is generated on its origin server. Since surrogates serve content authoritatively, objects obtained from them can always be considered fresh, and SHOULD contain a current Date header.

Passing non-current Date headers causes downstream caches to handle objects with an overly conservative freshness lifetime, if it is derived from either Cache-Control: max-age or some heuristic-based

freshness algorithms.

2.4.2 Interpretation of Proxy-Specific Information

Request headers such as Pragma: no-cache and some Cache-Control headers, if honored by surrogates, may cause excessive and unnecessary load on the master origin server.

2.4.3 Logging

Proxy-specific log formats may not be appropriate for use by a surrogate. In particular, master origin servers often log information such as the User-Agent and Referer presented by the

client.

Surrogates SHOULD be capable of logging such information, in a manner compatible with common origin server logs.

3. Specific Requirements

Requirements for a surrogate are the same as those for a gateway or proxy in HTTP/1.1[2], except as noted.

3.1 Protocol Version Interpretation

Implementations MUST satisfy the requirements of RFC 2145[5], including those behaviors specific to proxies.

3.2 Methods

A surrogate MUST NOT accept CONNECT requests, or forward them to the master origin server.

TRACE requests MAY be responded to as if max-forwards=0 were present, to keep the surrogate's relationship with the origin server private.

3.3 Status Codes

3.3.1 Redirections

Surrogates receiving redirections (301, 302 and 307 status codes) SHOULD resolve them and serve the resulting object to clients.

If surrogate-specific coherence is specified in a redirect, but not available for the resulting object, it SHOULD be applied to the object.

3.3.2 Error Conditions

Surrogates MUST NOT change the semantics of 4xx and 5xx series status codes obtained from origin servers. However, these responses MAY be cached for a short period.

401 Unauthorized status codes MAY be generated to propagate HTTP authentication; see "Working with Protocol Extensions".

Surrogates SHOULD send a 502 Bad Gateway error when surrogate-specific directives are incomplete, contradict themselves or don't parse correctly.

A 504 Gateway Timeout response SHOULD be sent under any of the following conditions:

- DNS failure when resolving the origin server
- no route to origin server
- refused connection to origin server
- connection timeout to origin server

However, a surrogate MAY be configured to use a cached resource, a different resource, or redirect to a different location under these conditions.

3.4 Cache Coherence and Correctness

The RECOMMENDED mechanism for assuring coherence on surrogates is use of Surrogate-Control request and response headers.

Surrogates MAY be configured to fall back to HTTP cache coherence (such as Expires and Cache-Control response headers), if surrogate-specific mechanisms are not available.

Surrogate origin servers MAY also be configured to use a heuristic freshness algorithm to ensure coherence if no other freshness information is available.

Because surrogates separate upstream and downstream coherence, they MAY also implement proprietary mechanisms for assuring coherence with the master origin server.

3.5 End-to-End Headers

Because a surrogate assumes the role of an origin server in downstream connections, the scope of end-to-end headers is changed. Although many headers can be propagated from the origin server, some must be changed in order to ensure protocol compliance, and others can be changed to enhance or optimize downstream connections.

3.5.1 Age

Surrogates MUST strip any Age header from responses before forwarding them to clients.

Surrogates MUST NOT add Age headers to responses.

Age headers SHOULD be used by surrogates in Age calculations, when determining coherence with the master origin server.

3.5.2 Cache-Control Request Header

Cache-Control headers in requests MUST NOT be honored by surrogates.

3.5.3 Cache-Control Response Header

By default, Cache-Control headers in responses from a master origin server MUST NOT be honored by surrogates, and MUST be forwarded to clients.

Surrogates SHOULD be able to be configured to honor Cache-Control response headers.

3.5.4 Date

Surrogate origin servers MUST serve a current Date header with each response; they MUST NOT serve a cached Date header.

3.5.5 ETag

If none are present, a surrogate MAY insert weak ETags as validators, if separate coherence with the master origin server has been established.

3.5.6 Expires

By default, Expires response headers SHOULD NOT be honored by surrogates, unless configured to do so. Surrogates MUST forward Expires headers to clients.

It has been observed that that if a Cache-Control: max-age response header is set, many origin servers will set a complimentary Expires: value, to duplicate the intended freshness effect for HTTP/1.0 clients. To accommodate this, surrogates SHOULD recalculate the Expires header to match the delta communicated in Cache-Control: max-age, but only if both are present in a response, and are equivalent.

Some older Web servers have been observed to set an Expires header

based on an offset from the Date, without setting a Cache-Control: max-age header. This is problematic, as it is difficult to distinguish these responses from those which wish to expire content at an absolute date. Surrogates MAY compensate for this by considering objects which specify an Expires without a Cache-Control: max-age directive stale when the Expires time is reached; however, this may have undesirable effects in some situations.

3.5.7 Host

Surrogate origin servers MUST replace any Host header in requests with the name of the appropriate master origin server before forwarding it.

3.5.8 Last-Modified

Last-Modified response headers MUST NOT be modified by a surrogate.

3.5.9 Pragma

Surrogate origin servers MUST NOT honor Pragma request directives.

3.5.10 Proxy-Authenticate

Surrogates MUST NOT include a Proxy-Authenticate header in responses to clients.

3.5.11 Proxy-Authorization

Surrogates MUST ignore Proxy-Authorization headers in requests from clients.

3.5.12 Server

Surrogates MAY set their own Server response header, replacing any present.

3.5.13 Via

Surrogates SHOULD append a Via header to requests, as outlined in RFC2616[2].

3.6 Surrogate-Control HTTP Headers

Surrogate-specific HTTP headers allow specification of metadata in requests or responses to the surrogate. These can be though of as analogies of cache-affecting headers such as Cache-Control.

Surrogate-Specific headers MUST be consumed before forwarding a request or response.

3.6.1 Surrogate-Control Request Header

Surrogate-Control request directives have similar semantics and effects as Cache-Control request headers. Defined directives are:

no-cache
: Has same meaning as a Cache-Control: max-age request directive to a proxy.

only-if-cached
: Has same meaning as a Cache-Control: only-if-cached request directive to a proxy.

3.6.2 Surrogate-Control Response Header

Surrogate-Control response directives have similar semantics and effects as Cache-Control response headers. Defined directives are:

max-age
: Has same meaning as a Cache-Control: max-age response directive to a proxy.

no-cache
: Has same meaning as a Cache-Control: no-cache response directive to a proxy.

must-revalidate
: Has same meaning as a Cache-Control: must-revalidate response directive to a proxy.

Surrogates SHOULD require some form of client authentication when honoring Surrogate-Control response directives.

3.7 Surrogate-Generated Headers

3.7.1 X-Forwarded-For Request Header

Surrogates SHOULD be capable of adding a header that denotes the client which requested the object.

3.7.2 X-Served-For Response Header

Surrogates MAY add a response header which denotes the name of the master origin server, if it is not obvious in the Request-URI, in order to enable third parties to identify the source of the content.

4. Working with Protocol Extensions

4.1 HTTP Authentication

Surrogates receiving responses with WWW-Authenticate headers MUST NOT serve them without assuring that the client has presented proper credentials.

HTTP Authentication may also be used to prevent access to the origin server by unauthorised clients, while allowing unauthenticated access to the objects through the surrogate. To accomplish this, a surrogate MAY be configured to send Authorization request headers, with a predetermined authentication realm.

5. Controlling Effects of Upstream Proxies

Surrogates SHOULD append appropriate Cache-Control and Pragma request headers to assure that any intermediate proxy/caches do not serve a response without validation on the master origin server.

6. Security Considerations

6.1 Surrogate to Origin Authentication and Security

Surrogates SHOULD allow use of SSL on the connection to the origin server, while serving objects unencrypted, to increase security between them.

They SHOULD also support at least one of the following authentication mechanisms for origin server access:

- Client-Side SSL Certificates

- HTTP Authentication into a specific realm (see "HTTP Authentication")
- Cookie-based authentication (using cookie value as shared secret)

6.2 Knowledge of Surrogate/Origin Relationship

It may or may not be necessary to hide the relationship between surrogates and origin servers, depending on the nature of their use.

Surrogates SHOULD allow configuration to accomplish this. Specifically, this includes all HTTP headers that identify responses as coming from a surrogate, TRACE requests, and error responses and warnings that identify the surrogate.

References

[1] Cooper, I., Melve, I. and G. Tomlinson, "Internet Web Replication and Caching Taxonomy", November 1999.

[2] Fielding, R., Gettys, J., Mogul, J. C., Frystyk, H., Masinter, L., Leach, P. and T. Berners-Lee, "Hypertext Transfer Protocol - HTTP/1.1", RFC 2616, June 1999.

[3] Berners-Lee, T., Fielding, R.T. and L. Masinter, "Uniform Resource Identifiers (URI): Generic Syntax", RFC 2396, August 1998.

[4] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", RFC 2119, March 1997.

[5] Fielding, R., Gettys, J., Mogul, J. C. and H. Frystyk, "Use and Intepretation of HTTP Version Numbers", RFC 2145, May 1997.

Author's Address

Mark Nottingham
Akamai Technologies
Suite 703, 1400 Fashion Island Bvld
San Mateo, CA 94404
US

EMail: mnot@akamai.com
URI: http://www.akamai.com/

Appendix A. Acknowledgements

The author gratefully acknowledges the contributions of: John Dilley, John Martin, Joel Wein, Peter Danzig, Chuck Neerdaels, and, David Karger.

Full Copyright Statement

Acknowledgement

Funding for the RFC editor function is currently provided by the Internet Society.