Network Working Group
Request for Comments: 2296
Category: Experimental

K. Holtman
TUE
A. Mutz
Hewlett-Packard

March 1998

HTTP Remote Variant Selection Algorithm -- RVSA/1.0

Status of this Memo

This memo defines an Experimental Protocol for the Internet community. It does not specify an Internet standard of any kind. Discussion and suggestions for improvement are requested. Distribution of this memo is unlimited.

Copyright Notice

ABSTRACT

HTTP allows web site authors to put multiple versions of the same information under a single URL. Transparent content negotiation is a mechanism for automatically selecting the best version when the URL is accessed. A remote variant selection algorithm can be used to speed up the transparent negotiation process. This document defines the remote variant selection algorithm with the version number 1.0.

TABLE OF CONTENTS

1 Introduction

HTTP allows web site authors to put multiple versions (variants) of the same information under a single URL. Transparent content negotiation [2] is a mechanism for automatically selecting the best variant when the URL is accessed. A remote variant selection algorithm can be used by a HTTP server to choose a best variant on behalf of a negotiating user agent. The use of a remote algorithm can speed up the transparent negotiation process by eliminating a request-response round trip.

This document defines the remote variant selection algorithm with the version number 1.0. The algorithm computes whether the Accept- headers in the request contain sufficient information to allow a choice, and if so, which variant must be chosen.

2 Terminology and notation

This specification uses the terminology and notation of the HTTP transparent content negotiation specification [2].

3 The remote variant selection algorithm

This section defines the remote variant selection algorithm with the version number 1.0. To implement this definition, a server MAY run any algorithm which gives equal results.

> Note: According to [2], servers are always free to return a list response instead of running a remote algorithm. Therefore, whenever a server may run a remote algorithm, it may also run a partial implementation of the algorithm, provided that the partial implementation always returns List_response when it cannot compute the real result.

3.1 Input

The algorithm is always run for a particular request on a particular transparently negotiable resource. It takes the following information as input.

1. The variant list of the resource, as present in the Alternates header of the resource.

2. (Partial) Information about capabilities and preferences of the user agent for this particular request, as given in the Accept- headers of the request.

If a fallback variant description

{"fallback.html"}

is present in the Alternates header, the algorithm MUST interpret it as the variant description

{"fallback.html" 0.000001}

The extremely low source quality value ensures that the fallback variant only gets chosen if all other options are exhausted.

3.2 Output

As its output, the remote variant selection algorithm and will yield the appropriate action to be performed. There are two possibilities:

Choice_response

The Accept- headers contain sufficient information to make a choice on behalf of the user agent possible, and the best variant MAY be returned in a choice response.

List_response

The Accept- headers do not contain sufficient information to make a choice on behalf of the user agent possible. A list response MUST be returned, allowing the user agent to make the choice itself.

3.3 Computing overall quality values

As a first step in the remote variant selection algorithm, the overall qualities of the individual variants in the list are computed.

The overall quality Q of a variant is the value

Q = round5(qs * qt * qc * ql * qf)

where round5 is a function which rounds a floating point value to 5 decimal places after the point, and where the factors qs, qt, qc, ql, and qf are determined as follows.

qs Is the source quality factor in the variant description.

qt The media type quality factor is 1 if there is no type attribute in the variant description, or if there is no Accept header in the request. Otherwise, it is the quality assigned by the Accept header to the media type in the type attribute.

Note: If a type is matched by none of the elements of an Accept header, the Accept header assigns the quality factor 0 to that type.

qc The charset quality factor is 1 if there is no charset attribute in the variant description, or if there is no Accept-Charset header in the request. Otherwise, the charset quality factor is the quality assigned by the Accept-Charset header to the charset in the charset attribute.

ql The language quality factor is 1 if there is no language attribute in the variant description, or if there is no Accept-Language header in the request. Otherwise, the language quality factor is the highest quality factor assigned by the Accept-Language header to any one of the languages listed in the language attribute.

qf The features quality factor is 1 if there is no features attribute in the variant description, or if there is no Accept-Features header in the request. Otherwise, it is the quality degradation factor for the features attribute, see section 6.4 of [2].

As an example, if a variant list contains the variant description

```
{"paper.html.en" 0.7 {type text/html} {language fr}}
```

and if the request contains the Accept- headers

```
Accept: text/html:q=1.0, */*:q=0.8
Accept-Language: en;q=1.0, fr;q=0.5
```

the remote variant selection algorithm will compute an overall quality for the variant as follows:

```
{"paper.html.fr" 0.7 {type text/html} {language fr}}
                  |           |                 |
                  |           |                 |
                  V           V                 V
        round5 ( 0.7   *     1.0        *      0.5 ) = 0.35000
```

With the above Accept- headers, the complete variant list

```
{"paper.html.en" 0.9 {type text/html} {language en}},
{"paper.html.fr" 0.7 {type text/html} {language fr}},
{"paper.ps.en"   1.0 {type application/postscript} {language en}}
```

would yield the following computations:

```
         round5 ( qs  * qt  * qc  * ql  * qf  ) =   Q
                  ---   ---   ---   ---   ---     -------
   paper.html.en: 0.9 * 1.0 * 1.0 * 1.0 * 1.0   = 0.90000
   paper.html.fr: 0.7 * 1.0 * 1.0 * 0.5 * 1.0   = 0.35000
   paper.ps.en:   1.0 * 0.8 * 1.0 * 1.0 * 1.0   = 0.80000
```

3.4 Definite and speculative quality values

A computed overall quality value can be either definite or speculative. An overall quality value is definite if it was computed without using any wildcard characters '*' in the Accept- headers, and without the need to use the absence of a particular Accept- header. An overall quality value is speculative otherwise.

As an example, in the previous section, the quality values of paper.html.en and paper.html.fr are definite, and the quality value of paper.ps.en is speculative because the type application/postscript was matched to the range */*.

Definiteness can be defined more formally as follows. An overall quality value Q is definite if the same quality value Q can be computed after the request message is changed in the following way:

1. If an Accept, Accept-Charset, Accept-Language, or Accept-Features header is missing from the request, add this header with an empty field.

2. Delete any media ranges containing a wildcard character '*' from the Accept header. Delete any wildcard '*' from the Accept-Charset, Accept-Language, and Accept-Features headers.

As another example, the overall quality factor for the variant

```
{"blah.html" 1 {language en-gb} {features blebber [x y]}}
```

is 1 and definite with the Accept- headers

```
Accept-Language: en-gb, fr
Accept-Features: blebber, x, !y, *
```

and

```
Accept-Language: en, fr
Accept-Features: blebber, x, *
```

The overall quality factor is still 1, but speculative, with the Accept- headers

```
Accept-language: en-gb, fr
Accept-Features: blebber, !y, *
```

and

```
Accept-Language: fr, *
Accept-Features: blebber, x, !y, *
```

3.5 Determining the result

The best variant, as determined by the remote variant selection algorithm, is the one variant with the highest overall quality value, or, if there are multiple variants which share the highest overall quality, the first variant in the list with this value.

The end result of the remote variant selection algorithm is Choice_response if all of the following conditions are met

a. the overall quality value of the best variant is greater than 0

b. the overall quality value of the best variant is a definite quality value

c. the variant resource is a neighbor of the negotiable resource. This last condition exists to ensure that a security-related restriction on the generation of choice responses is met, see sections 10.2 and 14.2 of [2].

In all other cases, the end result is List_response.

The requirement for definiteness above affects the interpretation of Accept- headers in a dramatic way. For example, it causes the remote algorithm to interpret the header

```
Accept: image/gif;q=0.9, */*;q=1.0
```

as

‘I accept image/gif with a quality of 0.9, and assign quality

factors up to 1.0 to other media types. If this information is insufficient to make a choice on my behalf, do not make a choice but send the list of variants'.

Without the requirement, the interpretation would have been

'I accept image/gif with a quality of 0.9, and all other media types with a quality of 1.0'.

4 Use of the algorithm

This section discusses how user agents can use the remote algorithm in an optimal way. This section is not normative, it is included for informational purposes only.

4.1 Using quality factors to rank preferences

Using quality factors, a user agent can not only rank the elements within a particular Accept- header, it can also express precedence relations between the different Accept- headers. Consider for example the following variant list:

```
{"paper.english" 1.0 {language en} {charset ISO-8859-1}},
{"paper.greek"   1.0 {language el} {charset ISO-8859-7}}
```

and suppose that the user prefers "el" over "en", while the user agent can render "ISO-8859-1" with a higher quality than "ISO-8859-7". If the Accept- headers are

```
Accept-Language: gr, en;q=0.8
Accept-Charset: ISO-8859-1, ISO-8859-7;q=0.6, *
```

then the remote variant selection algorithm would choose the English variant, because this variant has the least overall quality degradation. But if the Accept- headers are

```
Accept-Language: gr, en;q=0.8
Accept-Charset: ISO-8859-1, ISO-8859-7;q=0.95, *
```

then the algorithm would choose the Greek variant. In general, the Accept- header with the biggest spread between its quality factors gets the highest precedence. If a user agent allows the user to set the quality factors for some headers, while other factors are hard-coded, it should use a low spread on the hard-coded factors and a high spread on the user-supplied factors, so that the user settings take precedence over the built-in settings.

4.2 Construction of short requests

In a request on a transparently negotiated resource, a user agent need not send a very long Accept- header, which lists all of its capabilities, to get optimal results. For example, instead of sending

```
Accept: image/gif;q=0.9, image/jpeg;q=0.8, image/png;q=1.0,
        image/tiff;q=0.5, image/ief;q=0.5, image/x-xbitmap;q=0.8,
        application/plugin1;q=1.0, application/plugin2;q=0.9
```

the user agent can send

```
Accept: image/gif;q=0.9, */*;q=1.0
```

It can send this short header without running the risk of getting a choice response with, say, an inferior image/tiff variant. For example, with the variant list

```
{"x.gif" 1.0 {type image/gif}}, {"x.tiff" 1.0 {type image/tiff}},
```

the remote algorithm will compute a definite overall quality of 0.9 for x.gif and a speculative overall quality value of 1.0 for x.tiff. As the best variant has a speculative quality value, the algorithm will not choose x.tiff, but return a list response, after which the selection algorithm of the user agent will correctly choose x.gif. The end result is the same as if the long Accept- header above had been sent.

Thus, user agents can vary the length of the Accept- headers to get an optimal tradeoff between the speed with which the first request is transmitted, and the chance that the remote algorithm has enough information to eliminate a second request.

4.2.1 Collapsing Accept- header elements

This section discusses how a long Accept- header which lists all capabilities and preferences can be safely made shorter. The remote variant selection algorithm is designed in such a way that it is always safe to shorten an Accept or Accept-Charset header by two taking two header elements ‘A;q=f’ and ‘B;q=g’ and replacing them by a single element ‘P;q=m’ where P is a wildcard pattern that matches both A and B, and m is the maximum of f and g. Some examples are

```
text/html;q=1.0, text/plain;q=0.8       -->    text/*;q=1.0
image/*;q=0.8, application/*;q=0.7      -->    */*;q=0.8

iso-8859-5;q=1.0, unicode-1-1;q=0.8     -->    *;q=1.0
```

Note that every ‘;q=1.0’ above is optional, and can be omitted:

```
iso-8859-7;q=0.6, *                      -->    *
```

For Accept-Language, it is safe to collapse all language ranges with the same primary tag into a wildcard:

```
en-us;q=0.9, en-gb;q=0.7, en;q=0.8, da  -->    *;q=0.9, da
```

It is also safe to collapse a language range into a wildcard, or to replace it by a wildcard, if its primary tag appears only once:

```
*;q=0.9, da                              -->    *
```

Finally, in the Accept-Features header, every feature expression can be collapsed into a wildcard, or replaced by a wildcard:

```
colordepth!=5, *                         -->    *
```

4.2.2 Omitting Accept- headers

According to the HTTP/1.1 specification [1], the complete absence of an Accept header from the request is equivalent to the presence of ‘Accept: */*’. Thus, if the Accept header is collapsed to ‘Accept: */*’, a user agent may omit it entirely. An Accept-Charset, Accept-Language, or Accept-Features header which only contains ‘*’ may also be omitted.

4.2.3 Dynamically lengthening requests

In general, a user agent capable of transparent content negotiation can send short requests by default. Some short Accept- headers could be included for the benefit of existing servers which use HTTP/1.0 style negotiation (see section 4.2 of [2]). An example is

```
GET /paper HTTP/1.1
Host: x.org
User-Agent: WuxtaWeb/2.4
Negotiate: 1.0
Accept-Language: en, *;q=0.9
```

If the Accept- headers included in such a default request are not suitable as input to the remote variant selection algorithm, the user agent can disable the algorithm by sending ‘Negotiate: trans’ instead of ‘Negotiate: 1.0’.

If the user agent discovers, though the receipt of a list or choice response, that a particular origin server contains transparently negotiated resources, it could dynamically lengthen future requests to this server, for example to

```
GET /paper/chapter1 HTTP/1.1
Host: x.org
User-Agent: WuxtaWeb/2.4
Negotiate: 1.0
Accept: text/html, application/postscript;q=0.8, */*
Accept-Language: en, fr;q=0.5, *;q=0.9
Accept-Features: tables, *
```

This will increase the chance that the remote variant selection algorithm will have sufficient information to choose on behalf of the user agent, thereby optimizing the negotiation process. A good strategy for dynamic extension would be to extend the headers with those media types, languages, charsets, and feature tags mentioned in the variant lists of past responses from the server.

4.3 Differences between the local and the remote algorithm

A user agent can only optimize content negotiation though the use of a remote algorithm if its local algorithm will generally make the same choice. If a user agent receives a choice response containing a variant X selected by the remote algorithm, while the local algorithm would have selected Y, the user agent has two options:

1. Retrieve Y in a subsequent request. This is sub-optimal because it takes time.

2. Display X anyway. This is sub-optimal because it makes the end result of the negotiation process dependent on factors that can randomly change. For the next request on the same resource, and intermediate proxy cache could return a list response, which would cause the local algorithm to choose and retrieve Y instead of X. Compared to a stable representation, a representation which randomly switches between X and Y (say, the version with and without frames) has a very low subjective quality for most users.

As both alternatives above are unattractive, a user agent should try to avoid the above situation altogether. The sections below discuss how this can be done.

4.3.1 Avoiding major differences

If the user agent enables the remote algorithm in this specification, it should generally use a local algorithm which closely resembles the remote algorithm. The algorithm should for example also use multiplication to combine quality factors. If the user agent combines quality factors by addition, it would be more advantageous to define a new remote variant selection algorithm, with a new major version number, for use by this agent.

4.3.2 Working around minor differences

Even if a local algorithm uses multiplication to combine quality factors, it could use an extended quality formulae like

```
Q = round5( qs * qt * qc * ql * qf ) * q_adjust
```

in order to account for special interdependencies between dimensions, which are due to limitations of the user agent. For example, if the user agent, for some reason, cannot handle the iso-8859-7 charset when rendering text/plain documents, the q_adjust factor would be 0 when the text/plain - iso-8859-7 combination is present in the variant description, and 1 otherwise.

By selectively withholding information from the remote variant selection algorithm, the user agent can ensure that the remote algorithm will never make a choice if the local q_adjust is less than 1. For example, to prevent the remote algorithm from ever returning a text/plain - iso-8859-7 choice response, the user agent should take care to never produce a request which exactly specifies the quality factors of both text/plain and iso-8859-7. The omission of either factor from a request will cause the overall quality value of any text/plain - iso-8859-7 variant to be speculative, and variants with speculative quality values can never be returned in a choice response.

In general, if the local q_adjust does not equal 1 for a particular combination X - Y - Z, then a remote choice can be prevented by always omitting at least one of the elements of the combination from the Accept- headers, and adding a suitable wildcard pattern to match the omitted element, if such a pattern is not already present.

5 Security and privacy considerations

This specification introduces no security and privacy considerations not already covered in [2]. See [2] for a discussion of privacy risks connected to the sending of Accept- headers.

6 Acknowledgments

Work on HTTP content negotiation has been done since at least 1993. The authors are unable to trace the origin of many of the ideas incorporated in this document. Many members of the HTTP working group have contributed to the negotiation model in this specification. The authors wish to thank the individuals who have commented on earlier versions of this document, including Brian Behlendorf, Daniel DuBois, Ted Hardie, Larry Masinter, and Roy T. Fielding.

7 References

[1] Fielding, R., Gettys, J., Mogul, J., Frystyk, H., and T. Berners-Lee, "Hypertext Transfer Protocol -- HTTP/1.1", RFC 2068, January 1997.

[2] Holtman, K., and A. Mutz, "Transparent Content Negotiation in HTTP", RFC 2295, March 1998.

8 Authors' Addresses

Koen Holtman
Technische Universiteit Eindhoven
Postbus 513
Kamer HG 6.57
5600 MB Eindhoven (The Netherlands)

EMail: koen@win.tue.nl

Andrew H. Mutz
Hewlett-Packard Company
1501 Page Mill Road 3U-3
Palo Alto CA 94304, USA

Fax: +1 415 857 4691
EMail: mutz@hpl.hp.com

9 Full Copyright Statement