Internet Engineering Task Force
INTERNET DRAFT

Yaron Y. Goland
Microsoft Corporation
November 9, 1999
Expires April 2000

Multicast and Unicast UDP HTTP Messages
<draft-goland-http-udp-01.txt>

Status of this Memo

Abstract

This document provides rules for encapsulating HTTP messages in Multicast and Unicast UDP packets to be sent within a single administrative scope. No provisions are made for guaranteeing delivery beyond re-broadcasting.

1. Introduction

This document provides rules for encapsulating HTTP messages in multicast and unicast UDP messages. No provisions are made for guaranteeing delivery beyond re-broadcasting.

This technology is motivated by applications such as SSDP where it is expected that messages which are primarily transmitted over TCP HTTP need to be transmitted over Multicast or Unicast UDP in extreme circumstances.

This document will not specify a mechanism suitable for replacing HTTP over TCP. Rather this document will define a limited mechanism only suitable for extreme circumstances where the use of TCP is impossible. Thus this mechanism will not have the robustness of functionality and congestion control provided by TCP. It is expected that in practice the mechanisms specified here in will only be used as a means to get to TCP based HTTP communications.

2. Changes

2.1. Since 00

Divided each section of the spec into three parts, problem definition, proposed solution and design rationale. When the spec is ready for standardization the problem definition and design rationale sections will be removed. Design rationale is presented in question/answer form because I have found that to be very effective in addressing design issues.

Clarified that a HTTPU/HTTPMU URI without an abs_path translates to "*" in the request-URI.

Added the "S" header to allow request and responses to be associated. Note that while clients aren't require to send out "S" headers servers are required to return them.

Got rid of MM. The lower bound is always 0.

The introduction of the "S" makes proxying and caching possible so the sections on those topics have been expanded, but they should be considered experimental at best.

3. Terminology

Since this document describes a set of extensions to the HTTP/1.1 protocol, the augmented BNF used herein to describe protocol elements is exactly the same as described in section 2.1 of [RFC2616]. Since this augmented BNF uses the basic production rules provided in section 2.2 of [RFC2616], these rules apply to this

document as well.

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 [RFC2119].

4. HTTPU URL

4.1. Problem Definition

A mechanism is needed to allow for communications that are to be sent over Unicast UDP HTTP to be identified in the URI namespace.

4.2. Proposed Solution

The HTTPU URL specifies that the HTTP request is to be sent over unicast UDP according to the rules laid out in this document.

HTTPU_URL = "HTTPU:" "//" host [":" port] [abs_path ["?" query]]

The BNF productions host, port and abs_path are defined in [RFC2616].

The syntax of the HTTPU URL is to be processed identically to the HTTP URL with the exception of the transport.

One MUST NOT assume that if a HTTP, HTTPU or HTTPMU URL are identical in all ways save the protocol that they necessarily point to the same resource.

4.3. Design Rationale

4.3.1. Why would we ever need a HTTPU/HTTPMU URL?

Imagine one wants to tell a system to send responses over HTTPU. How would one express this? If one uses a HTTP URL there is no way for the system to understand that you really meant HTTPU.

5. HTTPMU URL

5.1. Problem Definition

A mechanism is needed to allow for communications that are to be sent over Multicast UDP HTTP to be identified in the URI namespace.

5.2. Proposed Solution

The HTTPMU URL specifies that the HTTP request that HTTP request is to be sent over multicast UDP according to the rules laid out in this document.

HTTPMU_URL = "HTTPMU:" "//" host [":" port] [abs_path ["?" query]]

The BNF productions host, port and abs_path are defined in [RFC2616].

The syntax of the HTTPMU URL is to be processed identically to the HTTP URL with the exception of the transport.

One MUST NOT assume that if a HTTP, HTTPU or HTTPMU URL are identical in all ways save the protocol that they necessarily point to the same resource.

If a HTTPMU URL does not have an abs_path element then when the HTTP UDP multicast request is made the request-URI MUST be "*".

For example, HTTPU://www.foo.com would translate into a request-URI of "*". Note, however, that HTTPU://www.foo.com/ would translate into a request-URI of "/".

5.3. Design Rationale

5.3.1. In the HTTPMU URL a request such as http://www.foo.com is translated to a "*" in the request-URI rather than a "/", why isn't the same the case for HTTPU?

A HTTPU request is a point-to-point request. There is one sender and one receiver. Thus the semantics of the URL are identical to HTTP with the exception of the transport.

In HTTPMU a request, generally, is going to many receivers. The way to indicate this on a HTTPMU request is by using the URI "*". Since using "*" is probably the single most common way to send a HTTPMU

request there needed to be a way to indicate that the request-URI should be "*". There is no way to do that today with a HTTP URL. Therefore a mechanism had to be added.

As a side note, one could send a point-to-point request of HTTPMU. One need only put a particular request-URI in the request. Only the resource matching that request-URI will respond.

6. Unicast UDP HTTP Messages

6.1. Problem Definition

A mechanism is needed to send HTTP messages over the unicast UDP transport.

6.2. Proposed Solution

HTTP messages sent over unicast UDP function identically to HTTP messages sent over TCP as defined in [RFC2616] except as specified below.

For brevity's sake HTTP messages sent over unicast UDP will be referred to as HTTPU messages.

HTTPU messages MUST fit entirely in a single UDP message. If a HTTPU message can not be fit into a single UDP message then it MUST NOT be sent using unicast UDP. Incomplete HTTPU messages SHOULD be ignored.

The request-URI of a HTTPU message MUST always be fully qualified.

A single unicast UDP message MUST only contain a single HTTPU message.

A HTTPU request without a "S" header MUST NOT be responded to. When responding to a HTTPU request with a "S" header the rules for the

proper handling of "S" headers, as specified in section 11.3 MUST be followed.

6.3. Design Rationale

See section 11.3.3 for the design rationale of the "S" header.

6.3.1. Why can't a single HTTP message be sent over multiple UDP messages?

The ability to send unlimited size messages across the Internet is one of the key features of TCP. The goal of this paper is not to re-invent TCP but rather to provide a very simple emergency back up HTTP system that can leverage UDP where TCP can not be used. As such features to allow a single HTTP message to span multiple UDP messages is not provided.

6.3.2. Why are request-URIs sent over HTTPU required to be fully qualified?

A relative URI in a HTTP message is assumed to be relative to a HTTP URL. However this would clearly be inappropriate for a HTTPU or HTTPMU message. The easiest solution would be to simply state that a relative URI is relative to the type of message it was sent in. But one of the unstated (but now stated) goals of this draft is to allow current HTTP message processors to be able to happily munch on HTTPU/HTTPMU messages and this would cause a change to those processors. Besides, relative URIs were always wacky, a left over from the early days of HTTP.
The cost of this simplification is that you repeat the host information, once in the URI and once in the host header. Eventually the host header will go away and we will all use fully qualified URIs. But again, taking out the host header would make a lot of existing HTTP message munchers very unhappy.

6.3.3. Why is the requirement for ignoring incomplete HTTPU messages a SHOULD instead of a MUST?

Some systems use a lot of redundant data or have good mechanisms for handling partial data. As such they could actually do something intelligent with a partial message. A SHOULD allows them to do this while still making it clear that in the majority case partial HTTPU/HTTPMU messages are going to get thrown out.

6.3.4. Why aren't multiple HTTP messages allowed into a single UDP message if they will fit?

It was easier to ban it and it didn't seem to buy us much. It was especially worrying because it would start to convince people that they could actually order their UDP requests in a pipelinesque manner. It was easier to just keep things simple and ban it.

6.3.5. Why aren't we allowed to leave off content-lengths if only a single HTTPU message is allowed in a UDP message?

In general we try to only change from RFC 2616 when we are forced to. Although including a content-length is annoying it makes it easy to use HTTP/1.1 message parsing/generating systems with this spec.

6.3.6. Why might a HTTPU message choose to not have a "S" header thus making it impossible to respond to it?

Leaving off the "S" header would be useful for throw away events. In systems with a high event rate it is usually easier to just throw away an event rather than re-sending it. As such there is no real benefit to confirming that the event was received since it won't be resent if it wasn't received.

6.3.7. Why isn't the mx header used on HTTPU messages?

As HTTPU messages are point-to-point there will be exactly one response. Mx is only useful in cases, such as HTTPMU requests, where there can be many potential responses from numerous different clients. Mx helps to prevent the client from getting creamed with responses.

6.3.8. Can I send 1xx responses over HTTPU?

Yes. Error handling is identical to RFC 2616.

7. Multicast UDP HTTP Requests

7.1. Problem Definition

A mechanism is needed to send HTTP messages over the multicast UDP transport.

7.2. Proposed Solution

HTTP messages sent over multicast UDP MUST obey all the requirements for HTTPU messages in addition to the requirements provided below.

For brevity's sake HTTP messages sent over multicast UDP will be referred to as HTTPMU messages.

Resources that support receiving multicast UDP HTTP requests MUST honor the mx header if included in the request.

Resources are required to generate a random number between 0 and mx

that represents the number of seconds the resource must wait before sending a response. This prevents all responses from being sent at once. HTTP clients SHOULD keep listening for responses for a reasonable delta of time after mx. That delta will be based on the type of network the request is being sent over. This means that if a

server cannot respond to a request before mx then there is little point in sending the response as the client will most likely not be listening for it.

When used with a multicast UDP HTTP request the "*" request-URI means "to everyone who is listening to this IP address and port."

A HTTPMU request without a mx header MUST NOT be responded to.

7.3. Design Rationale

7.3.1. Why is there a "delta" after the mx time when the client should still be listening?

So let's say the mx value is 5 seconds. The HTTP resource generates a number between 0 and 5 and gets 5. After 5 seconds of waiting the HTTP resource will send its response.

Now for some math:
0.5 seconds - Time it took the client's request to reach the HTTP resource.
5 seconds - Time the HTTP resource waited after receiving the message to respond, based on the mx value.
0.5 seconds - Time for the response to get back to the client.
Total time elapsed - 6 seconds

If the client only waits 5 seconds, the mx value, then they would have stopped listening for this response by the time it arrived. Hence the need for the delta.

7.3.2. What should the "delta" after mx expires be?

Unfortunately this is an impossible question to answer. How fast is your network? How far is the message going? Is there any congestion? In general delta values will be set based on a combination of heuristics and application necessity. That is, if you are displaying information to a user any data that comes in after 20 or 30 seconds

is probably too late.

7.3.3. When would a HTTPMU request not be responded to?

When a HTTP resource is making a general announcement, such as "I am here", it generally isn't useful to have everyone respond confirming they received the message. This is especially the case given that the HTTP resource probably doesn't know who should have received the announcement so the absence of a HTTP client in the responses wouldn't be meaningful.

7.3.4. Why do we require the mx header on HTTPMU requests that are to be responded to?

This is to prevent overloading the HTTP resource. If all the HTTP clients responded simultaneously the resource would probably loose most of the responses as its UDP buffer overflowed.

8. Retrying Requests

8.1. Problem Definition

UDP is an unreliable transport with no failure indicators, as such some mechanism is needed to reasonably increase the chance that a HTTPU/HTTPMU message will be delivered.

8.2. Proposed Solution

UDP is an inherently unreliable transport and subject to routers dropping packets without notice. Applications requiring delivery guarantees SHOULD NOT use HTTPU or HTTPMU.

In order to increase the probability that a HTTPU or HTTPMU message is delivered the message MAY be repeated several times.

In order to prevent the network from being flooded a message SHOULD NOT be repeated more than MAX_RETRIES time. A random period of time between 0 and MAX_RETRY_INTERVAL SHOULD be selected between each retry to determine how long to wait before issuing the retry.

8.3. Design Rationale

8.3.1. Why is the requirement "applications requiring delivery guarantees should not use HTTPU or HTTPMU" only a SHOULD and not a MUST?

Because there might come a day when it makes sense to use HTTPU or HTTPMU for guaranteed delivery and there is no reason to completely ban the possibility.

8.3.2. Why is the requirement that a request not be repeated more than MAX_RETRIES times a SHOULD and not a MUST?

Local knowledge may make the limit unnecessary. For example, if one knew that the message was being delivered using a super reliable network then repeats are not necessary. Similarly if one knew that the network the requests were going through were particularly unreliable and assuming one had properly accounted for the effects of additional messages on that congestion, one might have a good reason to send more than MAX_RETRIES.

9. Caching UDP HTTP Requests

9.1. Problem Definition

Caching is a feature that has demonstrated its usefulness in HTTP, provisions need to be made so as to ensure that HTTPU/HTTPMU messages can be cached using a consistent algorithm.

9.2. Proposed Solution

[Ed. Note: Never having tried to actually build a HTTPU/HTTPMU generic cache we suspect there are some very serious gotchas here that we just haven't found yet. This section should definitely be treated as "under development."]

Caching rules for HTTPU/HTTPMU responses are no different than normal HTTP responses. HTTPU/HTTPMU responses are matched to their requests through the "S" header value.

9.3. Design Rationale

9.3.1. Wouldn't it be useful to be able to cache HTTPU/HTTPMU requests if they don't have responses?

Yes, it probably would. Especially if we are talking about a client side cache. It is probably worth investigating the use of cache control headers on requests for this very purpose.

10. Proxying UDP HTTP Requests

10.1. Problem Definition

For security or caching reasons it is sometimes necessary to place a proxy in a message path. Provisions need to be made so as to ensure that HTTPU/HTTPMU messages can be proxied.

10.2. Proposed Solution

[Ed. Note: This section should be considered experimental. No one has really had to design much less implement a HTTPU/HTTPMU proxy yet.]

All transport independent rules for proxying, such as length of time to cache a response, hop-by-hop header rules, etc. are the same for HTTPU/HTTPMU as they are for HTTP messages.

[Ed. Note: I'm not sure how far to go into the "transport independent rules". The RFC 2616 doesn't really call them out very well but I also don't want to have to re-write RFC 2616 spec inside this spec.]

The transport dependent rules, however, are different. For example, using TCP any pipelined messages are guaranteed to be delivered in order. There are no ordering guarantees of any form for HTTPU/HTTPMU proxies.

In general a proxy is required to forward a HTTPU/HTTPMU message exactly once. It SHOULD NOT repeat the message. Rather the client is expected to repeat the message and, as the proxy receives the repeats, they will be forwarded.

The proxy is only responsible for forwarding responses to requests that include a "S" header. As with HTTPU/HTTPMU requests, responses

SHOULD NOT be repeated.

Note that it is acceptable, if not encouraged, for proxies to analyze network conditions and determine the likelihood, on both incoming and outgoing connections, of UDP messages being dropped. If the likelihood is too high then it would be expected for the proxy, taking into consideration the possibility of making congestion even worse, to repeat requests and responses on its own. In a sense the proxy could be thought of as a signal regenerator. This is why the prohibition against repeating messages is a SHOULD NOT rather than a MUST NOT.

HTTPMU messages are sent with the assumption that the message will only be seen by the multicast address they were sent to. Thus when a proxy forwards the request it is expected to only do so to the appropriate multicast channel. Note, however, that proxies may act as multicast bridges.

Also note that proxied HTTPMU messages with a HTTPMU URL without an absolute path are to be treated as if they were sent to the specified multicast address with the request-URI "*".

If a HTTPMU request is sent with a host that does not resolve to a multicast address then the request MUST be rejected with a 400 Bad Request error.

There is no requirement that a HTTPU proxy support HTTPMU or visa versa.

10.3. Design Rationale

10.3.1. Why would anyone proxy HTTPMU requests?

Proxying HTTPMU requests can be a neat way to create virtual multicast channels. Just hook a bunch of proxies together with unicast connections and tell the proxies' users that they are all on the same multicast scope.

11. HTTP Headers

11.1. AL Header

11.1.1. Problem Definition

There are many instances in which a system needs to provide location information using multiple URIs. The Location header only allows a single URI. Therefore a mechanism is needed to allow multiple location URIs to be returned.

11.1.2. Proposed Solution

AL = "AL" ":" 1*("<" AbsoluteURI ">") ; AbsoluteURI is defined in section 3.2.1 of [RFC2616]

The AL header is an extension of the Location header whose semantics are the same as the Location header. That is, the AL header allows one to return multiple locations where as the Location header allows one to return only one. The contents of an AL header are ordered. If both a Location header and an AL header are included in the same request then the URI in the location header is to be treated as if it were the first entry in the AL header. The AL header MAY be used by itself but implementers should be aware that existing systems will ignore the header.

11.1.3. Design Rationale

11.1.3.1. Why not just fix the BNF for the location header?

This is tempting but the goal of maintaining compatibility with RFC 2616's message format overrides the usefulness of this solution.

11.2. mx Request Header

11.2.1. Problem Definition

A mechanism is needed to ensure that responses to HTTPMU requests do not come at a rate greater than the requestor can handle.

11.2.2. Proposed Solution

[Ed. Note: We need to put in a max for this, at least a number after which the client isnÆt required to respond. 32 bit integer seconds sounds like overkill.]

```
mx = "mx" ":" Integer
Integer = First_digit *More_digits
First_digit = "1" | "2" | "3" | "4" | "5" | "6" | "7" | "8" | "9"
More_digits = "0" | First_digit
```

The value of the mx header indicates the maximum number of seconds that a multicast UDP HTTP resource MUST wait before it sends a

response stimulated by a multicast request.

HTTP resources MAY treat any mx header value greater than MX_MAX as being equal to MX_MAX.

11.2.3. Design Rationale

11.2.3.1. Why is mx in seconds?

In practice wait periods shorter than a second proved useless and longer proved too coarse. Of course as faster networks get deployed finer grain times would be useful but we need a compromise measurement that will meet everyone's needs, seconds seem to do that quite well.

11.2.3.2. Couldn't mx still overload the requestor if there are too many responders?

Absolutely. If there are a 100,000 clients that want to respond even pushing them over 30 seconds on a 10 Mbps link is still going to blow both the client and the network away. However the only way to prevent these sorts of situations is to know the current available network bandwidth and the total number of likely responders ahead of time. Both generally prove between difficult to impossible to figure out. So we are left with heuristics and the mx header.

11.3. S General Header

11.3.1. Problem Definition

A mechanism is needed to associated HTTPU/HTTPMU requests with responses as UDP does not have any connection semantics.

11.3.2. Proposed Solution

S = "S" ":" AbsoluteURI

The S header is a URI that is unique across the entire URI namespace for all time. When a "S" header is sent on a HTTPU/HTTPMU request it MUST be returned, with the same value, on the response.

If a client receives multiple responses with the same "S" header

then the client MAY assume that all the responses are from the same source and in response to the same request. If the messages differ from each other then the client MAY either throw all the responses away or randomly choose one to honor.

[Ed. Note: Ipv4 guarantees that the minimum MTU is 512 bytes or so long. The UUID URI takes 41 bytes if you don't add an extension element plus 5 extra characters for header over giving 46 characters. Assuming that the UUID is the minimum practical mechanism to guarantee globally unique messages this means that about 9% of every message is eaten up just by the S header. This is a lot. On the other hand most systems do much better than 512 bytes and Ipv6 requires (if memory serves) 4k that reduces the overhead to 1%. Note that UDP messages are still 64k long but I know lots of folks will want to optimize for a single UDP packet.

On the other hand, having a universally unique S header means that the algorithm for handling headers is very easy - if you see the same S value it is the same message. No worrying about rap arounds, time windows, or anything else. This is very appealing.]

11.3.3. Design Rationale

11.3.3.1. Why do we need the "S" header?

Without a "S" header the only way to match requests with responses is to ensure that there is enough information on the response to know what request it was intended to answer. Even in that case it is still possible to confuse which request a response goes to if it does not have the equivalent of a "S" header.

11.3.3.2. Couldn't the "S" header be used as a cookie?

No, "S" headers are sent out by clients and returned by servers. Cookies are sent out by servers and returned by clients.

11.3.3.3. Why aren't "S" headers mandatory on all requests with a response?

Some systems don't need them.

11.3.3.4. Why aren't "S" headers guaranteed to be sequential so you could do ordering?

Because HTTPU/HTTPMU is not interested in ordering. If one wants ordering one should use TCP.

12. Security Considerations

[Ed. Note: Besides putting in a note that all the normal HTTP security considerations apply we need to put in a discussion of the problems associated with requests getting lost as well as over sized request problem. We also need to talk about the fact that requests can get randomly lost. We also need to discuss how one uses authentication over UDP. Specifically, that one needs to assume the challenge and send the response as part of the request.]

[Ed. Note: Talk about the danger of abusing S headers.]

13. Acknowledgements

Thanks to John Stracke for his excellent comments.

14. Constants

MAX_RETRIES - 3

MAX_RETRY_INTERVAL - 10 seconds

MAX_MX - 120 seconds

15. References

[RFC2119] S. Bradner. Key words for use in RFCs to Indicate Requirement Levels. RFC 2119, March 1997.

[RFC2616] R. Fielding, J. Gettys, J. C. Mogul, H. Frystyk, L. Masinter, P. Leach and T. Berners-Lee. Hypertext Transfer Protocol - HTTP/1.1. RFC 2616, November 1998.

16. Author's Address

Yaron Y. Goland
Microsoft Corporation
One Microsoft Way

Redmond, WA 98052

Email: yarong@microsoft.com

This document will expire in April 2000.