Embedded Binary HTTP (EBHTTP)
draft-tolle-core-ebhttp-00

Abstract

Embedded Binary HTTP (EBHTTP) is a binary-formatted, space-efficient, stateless encoding of the standard HTTP/1.1 protocol. EBHTTP is intended for transport of small named data items, such as sensor readings, between resource-constrained nodes.

Status of this Memo

Copyright Notice

Table of Contents

1. Introduction

HTTP/1.1 [RFC2616] is an application-level protocol for distributed, collaborative, hypermedia information systems, and is the underlying protocol of the World Wide Web. HTTP is a simple ASCII request/response protocol that runs over TCP/IP, in which a client sends a request to a server containing a request method, a URI identifying a resource to be operated upon, and an optional request body, and receives a response containing a status code and an optional response body.

Fielding, in [FieldingArch] suggests that the Web architecture succeeded because it has:

a low barrier to entry

a simple protocol design

extensibility

From the principles and constraints of the Web architecture, Fielding derived an architecture called Representational State Transfer (REST). REST can be described as a combination of several architectural styles:

client-server interaction

stateless requests

cacheable responses

uniform interface

layered system

The key abstractions within a RESTful design are:

resources (conceptual)

resource identifiers (URI)

representations of resources (documents, like HTML and XML)

representation metadata

resource metadata

control data

On the RESTful web, HTTP is the primary transfer protocol, but resources should exist apart from the exact protocol used to transfer them across the network.

The RESTful style originated in human interactions mediated through hypertext, but can also be used as the basis for a wide-area system for machine-to-machine communications. In machine-to-machine communications, the transfer protocol (HTTP) and resource naming system (URI) remains the same, but the representations of resources are typically defined in a machine-readable format, such as XML, coupled with a description of the actions to be taken in response to data embedded in the request or representation.

While most machine-to-machine communication systems can use HTTP directly, HTTP is less suitable for certain highly-constrained networks. When network bandwidth is limited, the human-readable ASCII messages used by HTTP can become a source of congestion. When networks are asymmetrical and high-latency, the TCP protocol underlying HTTP can reduce effective throughput. When unreliable one-way communication is acceptable, the request-response nature of HTTP can introduce unwanted overhead. When nodes are limited in storage and processing power, the code needed for generating and consuming HTTP messages can compete with the application code itself.

This document proposes a protocol called Embedded Binary HTTP (EBHTTP), which maintains the simplicity, RESTful design, and extensibility of standard HTTP while being more suitable for highly-constrained networks. EBHTTP replaces the ASCII HTTP messages with compact binary messages and replaces TCP with UDP (while still allowing TCP), and remains faithful enough to the particulars of HTTP to enable stateless, application-independent transcoding between the two protocols, as long as the original HTTP request fits within any limits defined by the EBHTTP protocol and any limits implemented by the EBHTTP node.

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 [RFC2119].

1.2. Basic Operation

EBHTTP can be used between any two hosts on the Internet, regardless of whether the hosts are constrained or not. An EBHTTP client makes a request to an EBHTTP server, which may return an EBHTTP response.

An EBHTTP translation proxy waits to accept requests from both HTTP and EBHTTP clients. When a HTTP request is received, the proxy makes an EBHTTP request on the client's behalf. When an EBHTTP request is received, the proxy makes a HTTP request on the client's behalf.

2. Message Format

2.1. Basic Message Format

An EBHTTP message has the following basic format:

```
 0                   1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|    Control    |    Method     |  URI Length   |  Body Length  |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|    URI...
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|    Body...
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

Control This byte is broken down into the following bit fields:

```
+-+-+-+-+-+-+-+-+
|  Ver  |H|R|   |
+-+-+-+-+-+-+-+-+
```

Ver This field is the version number of EBHTTP. This MUST be set to 1, until superseded by future revisions of this specification.

H This field is set to 1 when additional EBHTTP headers are present between the URI and the Body. When a message is received with this bit set, the receiver MUST either process or ignore every header in the message, before processing the Body.

R This field is set to 1 when a response is required. When a message is received with this bit set, the receiver MUST respond with an EBHTTP response message. When the bit is not set, the receiver SHOULD not send a response, but MAY still respond.

The remaining two bits are reserved for future use.

Method This byte represents either an encoded HTTP method or an encoded HTTP status code. The encodings of the methods are defined here:

```
typedef enum ebhttp_method {
  EBHTTP_GET = 1,
  EBHTTP_POST = 2,
  EBHTTP_PUT = 3,
  EBHTTP_DELETE = 4,
  EBHTTP_HEAD = 5,
  EBHTTP_OPTIONS = 6,
  EBHTTP_TRACE = 7,
  EBHTTP_CONNECT = 8,
  EBHTTP_SUBSCRIBE = 9,
  EBHTTP_UNSUBSCRIBE = 10,
  EBHTTP_NOTIFY = 11,
} ebhttp_method_t;
```

The encodings of the status codes are created by packing the status code class into the first 3 bits, and the status code value into the last 5 bits, as so:

```
typedef enum ebhttp_status {
  EBHTTP_200_OK = 2 << 5 | 00,
  EBHTTP_229_SUBSCRIPTION_SUCCEEDED = 2 << 5 | 29,
  EBHTTP_230_NOTIFICATION_ACKNOWLEDGED = 2 << 5 | 30,
  EBHTTP_231_SUBSCRIPTION_TERMINATED = 2 << 5 | 31,
  EBHTTP_400_BAD_REQUEST = 4 << 5 | 00,
  EBHTTP_404_NOT_FOUND = 4 << 5 | 04,
  EBHTTP_405_METHOD_NOT_ALLOWED = 4 << 5 | 05,
  EBHTTP_429_SUBSCRIPTION_FAILED = 4 << 5 | 29,
  EBHTTP_500_INTERNAL_SERVER_ERROR = 4 << 5 | 00,
  ... other status codes are constructed similarly ...
} ebhttp_status_t;
```

Because the encoding of the HTTP request methods always have 0's in their 3 most significant bits, and the encodings of the status codes do not, requests can be distinguished from responses solely by the contents of the Method field -- without requiring a separate bitfield.

URI Length This field specifies the length of the URI in bytes. URIs longer than 255 bytes MAY be included by setting this field to 255, and then storing the actual URI length into the first 4 bytes of the URI field as an unsigned 4-byte integer in network byte order. This length does not include the 4-byte integer

itself. EBHTTP message receivers MUST interpret this "extended length" mode when determining the true start of the URI within the message, but EBHTTP message receivers MAY return the status code 414 "Request-URI Too Large" if the specified length of the URL exceeds the capabilities of the host.

Body Length This field specifies the length of the body data in bytes. Bodies longer than 255 bytes MAY be included by setting this field to 255, and then storing the actual body length into the first 4 bytes of the Body field as an unsigned 4-byte integer in network byte order. This length does not include the 4-byte integer itself. EBHTTP message receivers MUST interpret this "extended length" mode when determining the true start of the Body within the message, but MAY return the status code 413 "Request Entity Too Large" if the specified length of the body exceeds the capabilities of the host.

URI This field contains the text of the HTTP URI.

Body This field contains the contents of the HTTP message body.

2.2. Optional Header Format

Each HTTP header is encoded as a single EBHTTP optional header. Optional headers are placed after the end of the URI field, and before the start of Body field. They are packed back to back with no padding. Both requests and responses may contain optional headers, matching the HTTP standard.

Optional headers have the following format:

```
 0                   1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|     Type      |     Length    | Value ...
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

The Length byte MUST be interpreted in the same way as the URI length and Body Length bytes defined above. If a header Value longer than 255 bytes is specified, then the first 4 bytes of the header Value MUST be interpreted as an unsigned 4-byte integer in network byte order.

EBHTTP includes space for a 7-bit Type code, and defines an encoded value for each of the 57 standard HTTP request and response headers currently defined [TODO]. This leaves room to expand the space of encoded headers in the future.

For commonly-used header values, like MIME types, EBHTTP also defines a set of encoded values [TODO]. Headers containing the encoded value and headers containing the unencoded value must have separate type codes.

In addition to encoding the standard HTTP headers, EBHTTP supports encoded EBHTTP-only headers. One such header is "Request-ID", which contains a token that can be used to match requests with responses. This "Request-ID" header MUST be maintained in any responses to requests that contain it. EBHTTP implementations MUST process the "Request-ID" header.

If the host wishes to specify a HTTP header that is not currently encoded by EBHTTP, then the host specifies a Type of 255. The full, unencoded, ASCII HTTP header is then placed into the Value field.

Hosts MUST ignore any headers that they are unable to process.

3. Protocol Implementation

EBHTTP hosts MUST support UDP as a protocol for carrying EBHTTP messages.

EBHTTP requests are transmitted from clients to servers over UDP, and EBHTTP responses are transmitted back to clients, using the client's UDP source port.

EBHTTP responses are optional, as described above.

Multiple EBHTTP messages MAY be packed into a single UDP datagram, and EBHTTP servers MUST unpack these messages and treat them as separate requests. All the lengths are specified within the message, making this possible. EBHTTP message packing allows more efficient use of the network by reducing the number of redundant UDP and IP headers, and SHOULD be used in constrained networks.

Multiple EBHTTP responses MAY also be packed into a single UDP message. EBHTTP clients MUST unpack these responses and treat them the same way as responses contained in individual datagrams. The "Request-ID" header SHOULD be used to match requests and responses.

A single EBHTTP message MUST NOT span multiple UDP datagrams. EBHTTP does not support fragmentation. If EBHTTP needs to carry messages larger than a single datagram, then TCP MUST be used.

EBHTTP hosts MAY run EBHTTP over TCP.

EBHTTP requests and responses MUST be packed back-to-back into the TCP bytestream, with no padding. Any number of requests and responses may be sent over a single TCP connection.

4. Caching

EBHTTP implementations should follow the HTTP caching behavior described in [RFC2616]. TODO...

5. Proxies

EBHTTP implementations should support the HTTP proxy behavior, where necessary. TODO...

6. Publish/Subscribe

Publish/Subscribe functionality may be implemented over EBHTTP using application-defined message formats.

In addition, EBHTTP supports the HTTP-level publish/subscription mechanisms described in the General Event Notification Architecture (GENA) Internet-Draft [cohen-gena-p-base]. These mechanisms include new HTTP methods for SUBSCRIBE, UNSUBSCRIBE, NOTIFY, and POLL, and new headers for conveying subscription IDs, callback URLs, and subscription lifetimes.

7. Resource Naming

EBHTTP carries full HTTP URLs, but URLs for services intended for use within constrained networks should be as compact as possible. By using short letter and number codes, meaningful resource names can be compacted into only a few bytes.

8. Resource Encoding

EBHTTP makes no statement about the encoding of the actual resources it can transport. However, because EBHTTP is intended for use in constrained networks, EBHTTP users should try to provide compact resources, and then use profile systems to automatically expand those compact resources into formats that are more usable on higher-end hosts.

9. Resource Discovery

As befits the RESTful style, resource discovery should begin by retrieving the base URL representing the service.

This resource should contain a machine readable representation of the URLs for the resources supported by the service (suggest using one of the Binary XML encodings, or a more compact encoding needed by the application). These resources may represent data or further indexes, depending on the definition of the service using EBHTTP.

This index resource may also be proactively sent to another server, which will then act as a discovery proxy on behalf of the originator.

Discovery of the network addresses of EBHTTP-speaking hosts can be performed via DNS-Service Discovery and Multicast DNS, as defined in [I-D.cheshire-dnsext-dns-sd] and [I-D.cheshire-dnsext-multicastdns].

TODO...

10. Security Considerations

HTTP includes several security mechanisms, including digest authentication for passwords and SSL for transport. These mechanisms should be supported in EBHTTP as well. Application layer protocols above EBHTTP may also include additional authentication, key exchange, and encryption techniques. TODO...

11. References

11.1. Normative References

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, March 1997.

[RFC2616] Fielding, R., Gettys, J., Mogul, J., Frystyk, H., Masinter, L., Leach, P., and T. Berners-Lee, "Hypertext Transfer Protocol -- HTTP/1.1", RFC 2616, June 1999.

11.2. Informative References

[FieldingArch]
Fielding, R., "Architectural Styles and the Design of Network-based Software Architectures", 2000.

[I-D.cheshire-dnsext-dns-sd]

Cheshire, S. and M. Krochmal, "DNS-Based Service Discovery", draft-cheshire-dnsext-dns-sd-06 (work in progress), March 2010.

[I-D.cheshire-dnsext-multicastdns]
Cheshire, S. and M. Krochmal, "Multicast DNS", draft-cheshire-dnsext-multicastdns-10 (work in progress), March 2010.

Author's Address

Gilman Tolle
Arch Rock Corporation
501 2nd St. Ste 410
San Francisco 94107
US

Phone: 415-692-0828
Email: gtolle@archrock.com
URI: http://www.archrock.com