

Network Working Group
Internet-Draft
Intended status: Standards Track
Expires: October 4, 2018

P. Hoffman
ICANN
P. McManus
Mozilla
April 02, 2018

DNS Queries over HTTPS draft-ietf-doh-dns-over-https-05

Abstract

This document describes how to run DNS service over HTTP using https:// URIs.

[[There is a repository for this draft at <https://github.com/dohwg/draft-ietf-doh-dns-over-https> [1]]].

Status of This Memo

This Internet-Draft is submitted in full conformance with the provisions of [BCP 78](#) and [BCP 79](#).

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF). Note that other groups may also distribute working documents as Internet-Drafts. The list of current Internet-Drafts is at <https://datatracker.ietf.org/drafts/current/>.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

This Internet-Draft will expire on October 4, 2018.

Copyright Notice

Copyright (c) 2018 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to [BCP 78](#) and the IETF Trust's Legal Provisions Relating to IETF Documents (<https://trustee.ietf.org/license-info>) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must include Simplified BSD License text as described in [Section 4](#).e of

the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.

Table of Contents

1.	Introduction	2
2.	Terminology	3
3.	Protocol Requirements	3
3.1.	Non-requirements	4
4.	The HTTP Request	4
4.1.	DNS Wire Format	5
4.2.	Examples	5
5.	The HTTP Response	6
5.1.	Example	8
6.	HTTP Integration	8
6.1.	Cache Interaction	8
6.2.	HTTP/2	9
6.3.	Server Push	9
6.4.	Content Negotiation	10
7.	IANA Considerations	10
7.1.	Registration of application/dns-udpwireformat Media Type	10
8.	Security Considerations	12
9.	Operational Considerations	13
10.	Acknowledgments	14
11.	References	14
11.1.	Normative References	14
11.2.	Informative References	15
11.3.	URIs	16
Appendix A.	Previous Work on DNS over HTTP or in Other Formats .	16
Authors' Addresses	17

[1.](#) Introduction

This document defines a specific protocol for sending DNS [[RFC1035](#)] queries and getting DNS responses over HTTP [[RFC7540](#)] using https:// (and therefore TLS [[RFC5246](#)] security for integrity and confidentiality). Each DNS query-response pair is mapped into a HTTP request-response pair.

The described approach is more than a tunnel over HTTP. It establishes default media formatting types for requests and responses but uses normal HTTP content negotiation mechanisms for selecting alternatives that endpoints may prefer in anticipation of serving new use cases. In addition to this media type negotiation, it aligns itself with HTTP features such as caching, redirection, proxying, authentication, and compression.

The integration with HTTP provides a transport suitable for both traditional DNS clients and native web applications seeking access to the DNS.

Two primary uses cases were considered during this protocol's development. They included preventing on-path devices from interfering with DNS operations and allowing web applications to access DNS information via existing browser APIs in a safe way consistent with Cross Origin Resource Sharing (CORS) [[CORS](#)]. There are certainly other uses for this work.

2. Terminology

A server that supports this protocol is called a "DNS API server" to differentiate it from a "DNS server" (one that uses the regular DNS protocol). Similarly, a client that supports this protocol is called a "DNS API client".

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "NOT RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in [BCP 14](#), [RFC8174](#) [[RFC8174](#)] when, and only when, they appear in all capitals, as shown here.

3. Protocol Requirements

The protocol described here bases its design on the following protocol requirements:

- o The protocol must use normal HTTP semantics.
- o The queries and responses must be able to be flexible enough to express every DNS query that would normally be sent in DNS over UDP (including queries and responses that use DNS extensions, but not those that require multiple responses).
- o The protocol must permit the addition of new formats for DNS queries and responses.
- o The protocol must ensure interoperability by specifying a single format for requests and responses that is mandatory to implement. That format must be able to support future modifications to the DNS protocol including the inclusion of one or more EDNS options (including those not yet defined).
- o The protocol must use a secure transport that meets the requirements for HTTPS.

3.1. Non-requirements

- o Supporting network-specific DNS64 [[RFC6147](#)]
- o Supporting other network-specific inferences from plaintext DNS queries
- o Supporting insecure HTTP

4. The HTTP Request

A DNS API client encodes a single DNS query into an HTTP request using either the HTTP GET or POST method and the other requirements of this section. The DNS API server defines the URI used by the request through the use of a URI Template [[RFC6570](#)]. Configuration and discovery of the URI Template is done out of band from this protocol.

The URI template defined in this document is processed without any variables for requests using POST, and with the single variable "dns" for requests using GET. The value of the dns parameter is the content of the request (as described in [Section 4.1](#)), encoded with base64url [[RFC4648](#)].

Future specifications for new media types MUST define the variables used for URI Template processing with this protocol.

DNS API servers MUST implement both the POST and GET methods.

When using the POST method the DNS query is included as the message body of the HTTP request and the Content-Type request header indicates the media type of the message. POST-ed requests are smaller than their GET equivalents.

Using the GET method is friendlier to many HTTP cache implementations.

The DNS API client SHOULD include an HTTP "Accept" request header to indicate what type of content can be understood in response. Irrespective of the value of the Accept request header, the client MUST be prepared to process "application/dns-udpwireformat" [Section 4.1](#) responses but MAY also process any other type it receives.

In order to maximize cache friendliness, DNS API clients using media formats that include DNS ID, such as application/dns-udpwireformat, SHOULD use a DNS ID of 0 in every DNS request. HTTP correlates request and response, thus eliminating the need for the ID in a media

type such as `application/dns-udpwireformat` and the use of a varying DNS ID can cause semantically equivalent DNS queries to be cached separately.

DNS API clients can use HTTP/2 padding and compression in the same way that other HTTP/2 clients use (or don't use) them.

4.1. DNS Wire Format

The data payload is the DNS on-the-wire format defined in [RFC1035]. The format is for DNS over UDP. Note that this is different than the wire format used in [RFC7858]. Also note that while [RFC1035] says "Messages carried by UDP are restricted to 512 bytes", that was later updated by [RFC6891], and this protocol allows DNS on-the-wire format payloads of any size.

When using the GET method, the data payload MUST be encoded with `base64url` [RFC4648] and then provided as a variable named "dns" to the URI Template expansion. Padding characters for `base64url` MUST NOT be included.

When using the POST method, the data payload MUST NOT be encoded and is used directly as the HTTP message body.

DNS API clients using the DNS wire format MAY have one or more EDNS options [RFC6891] in the request.

The media type is `"application/dns-udpwireformat"`.

4.2. Examples

These examples use HTTP/2 style formatting from [RFC7540].

These examples use a DNS API service with a URI Template of `"https://dnsserver.example.net/dns-query{?dns}"` to resolve IN A records.

The requests are represented as `application/dns-udpwireformat` typed bodies.

The first example request uses GET to request `www.example.com`

```
:method = GET
:scheme = https
:authority = dnsserver.example.net
:path = /dns-query?dns=AAABAAABAAAAAAA3d3dwdleGFtcGx1A2NvbQAAAQAB
accept = application/dns-udpwireformat
```


The same DNS query for `www.example.com`, using the POST method would be:

```
:method = POST
:scheme = https
:authority = dnsserver.example.net
:path = /dns-query
accept = application/dns-udpwireformat
content-type = application/dns-udpwireformat
content-length = 33
```

<33 bytes represented by the following hex encoding>

```
00 00 01 00 00 01 00 00 00 00 00 00 03 77 77 77
07 65 78 61 6d 70 6c 65 03 63 6f 6d 00 00 01 00
01
```

Finally, a GET based query for `a.62characterlabel-makes-base64url-distinct-from-standard-base64.example.com` is shown as an example to emphasize that the encoding alphabet of `base64url` is different than regular `base64` and that padding is omitted.

The DNS query is 94 bytes represented by the following hex encoding

```
00 00 01 00 00 01 00 00 00 00 00 00 01 61 3e 36
32 63 68 61 72 61 63 74 65 72 6c 61 62 65 6c 2d
6d 61 6b 65 73 2d 62 61 73 65 36 34 75 72 6c 2d
64 69 73 74 69 6e 63 74 2d 66 72 6f 6d 2d 73 74
61 6e 64 61 72 64 2d 62 61 73 65 36 34 07 65 78
61 6d 70 6c 65 03 63 6f 6d 00 00 01 00 01
```

```
:method = GET
:scheme = https
:authority = dnsserver.example.net
:path = /dns-query? (no space or CR)
      dns=AAABAAABAAAAAAWE-NjJjaGFyYWN0ZXJsYWJl (no space or CR)
      bC1tYWtlcy1iYXNlNjR1cmwtZGlzdGluY3QtZnJvbS1z (no space or CR)
      dGFuZGFyZC1iYXNlNjQHZXhhbXBsZQNjb20AAAEAAQ
accept = application/dns-udpwireformat
```

5. The HTTP Response

An HTTP response with a 2xx status code ([\[RFC7231\] Section 6.3](#)) indicates a valid DNS response to the query made in the HTTP request. A valid DNS response includes both success and failure responses. For example, a DNS failure response such as `SERVFAIL` or `NXDOMAIN` will be the message in a successful 2xx HTTP response even though there was a failure at the DNS layer. Responses with non-successful HTTP

status codes do not contain DNS answers to the question in the corresponding request. Some of these non-successful HTTP responses (e.g., redirects or authentication failures) could allow clients to make new requests to satisfy the original question.

Different response media types will provide more or less information from a DNS response. For example, one response type might include the information from the DNS header bytes while another might omit it. The amount and type of information that a media type gives is solely up to the format, and not defined in this protocol.

At the time this is published, the response types are works in progress. The only response type defined in this document is "application/dns-udpwireformat", but it is possible that other response formats will be defined in the future.

The DNS response for "application/dns-udpwireformat" in [Section 4.1](#) MAY have one or more EDNS options, depending on the extension definition of the extensions given in the DNS request.

Each DNS request-response pair is matched to one HTTP request-response pair. The responses may be processed and transported in any order using HTTP's multi-streaming functionality ([\[RFC7540\]](#) [Section 5](#)}).

The Answer section of a DNS response can contain zero or more RRsets. (RRsets are defined in [\[RFC7719\]](#).) According to [\[RFC2181\]](#), each resource record in an RRset has Time To Live (TTL) freshness information. Different RRsets in the Answer section can have different TTLs, although it is only possible for the HTTP response to have a single freshness lifetime. The HTTP response freshness lifetime ([\[RFC7234\]](#) [Section 4.2](#)) should be coordinated with the RRset with the smallest TTL in the Answer section of the response. Specifically, the HTTP freshness lifetime SHOULD be set to expire at the same time any of the DNS resource records in the Answer section reach a 0 TTL. The response freshness lifetime MUST NOT be greater than that indicated by the DNS resource record with the smallest TTL in the response.

If the DNS response has no records in the Answer section, and the DNS response has an SOA record in the Authority section, the response freshness lifetime MUST NOT be greater than the MINIMUM field from that SOA record. (See [\[RFC2308\]](#).) Otherwise, the HTTP response MUST set a freshness lifetime ([\[RFC7234\]](#) [Section 4.2](#)) of 0 by using a mechanism such as "Cache-Control: no-cache" ([\[RFC7234\]](#) [Section 5.2.1.4](#)).

A DNS API client that receives a response without an explicit freshness lifetime MUST NOT assign that response a heuristic freshness ([\[RFC7234\] Section 4.2.2.](#)) greater than that indicated by the DNS Record with the smallest TTL in the response.

A DNS API server MUST be able to process application/dns-udpwireformat request messages.

A DNS API server SHOULD respond with HTTP status code 415 (Unsupported Media Type) upon receiving a media type it is unable to process.

This document does not change the definition of any HTTP response codes or otherwise proscribe their use.

5.1. Example

This is an example response for a query for the IN A records for "www.example.com" with recursion turned on. The response bears one record with an address of 192.0.2.1 and a TTL of 128 seconds.

```
:status = 200
content-type = application/dns-udpwireformat
content-length = 64
cache-control = max-age=128
```

<64 bytes represented by the following hex encoding>

```
00 00 81 80 00 01 00 01 00 00 00 00 03 77 77 77
07 65 78 61 6d 70 6c 65 03 63 6f 6d 00 00 01 00
01 03 77 77 77 07 65 78 61 6d 70 6c 65 03 63 6f
6d 00 00 01 00 01 00 00 00 80 00 04 c0 00 02 01
```

6. HTTP Integration

This protocol MUST be used with the https scheme URI [\[RFC7230\]](#).

6.1. Cache Interaction

A DNS API client may utilize a hierarchy of caches that include both HTTP and DNS specific caches. HTTP cache entries may be bypassed with HTTP mechanisms such as the "Cache-Control no-cache" directive; however DNS caches do not have a similar mechanism.

A DOH response that was previously stored in an HTTP cache will contain the [\[RFC7234\]](#) Age response header indicating the elapsed time between when the entry was placed in the HTTP cache and the current DOH response. DNS API clients should subtract this time from the DNS TTL if they are re-sharing the information in a non HTTP context

(e.g., their own DNS cache) to determine the remaining time to live of the DNS record.

HTTP revalidation (e.g., via If-None-Match request headers) of cached DNS information may be of limited value to DOH as revalidation provides only a bandwidth benefit and DNS transactions are normally latency bound. Furthermore, the HTTP response headers that enable revalidation (such as "Last-Modified" and "Etag") are often fairly large when compared to the overall DNS response size, and have a variable nature that creates constant pressure on the HTTP/2 compression dictionary [[RFC7541](#)]. Other types of DNS data, such as zone transfers, may be larger and benefit more from revalidation. DNS API servers may wish to consider whether providing these validation enabling response headers is worthwhile.

The stale-while-revalidate and stale-if-error cache control directives may be well suited to a DOH implementation when allowed by server policy. Those mechanisms allow a client, at the server's discretion, to reuse a cache entry that is no longer fresh under some extenuating circumstances defined in [[RFC5861](#)].

All HTTP servers, including DNS API servers, need to consider cache interaction when they generate responses that are not globally valid. For instance, if a DNS API server customized a response based on the client's identity then it would not want to globally allow reuse of that response. This could be accomplished through a variety of HTTP techniques such as a Cache-Control max-age of 0, or perhaps by the Vary response header.

[6.2.](#) HTTP/2

The minimum version of HTTP used by DOH SHOULD be HTTP/2 [[RFC7540](#)].

The messages in classic UDP based DNS [[RFC1035](#)] are inherently unordered and have low overhead. A competitive HTTP transport needs to support reordering, parallelism, priority, and header compression to achieve similar performance. Those features were introduced to HTTP in HTTP/2 [[RFC7540](#)]. Earlier versions of HTTP are capable of conveying the semantic requirements of DOH but may result in very poor performance for many use cases.

[6.3.](#) Server Push

Before using DOH response data for DNS resolution, the client MUST establish that the HTTP request URI is a trusted service for the DOH query. For HTTP requests initiated by the DNS API client this trust is implicit in the selection of URI. For HTTP server push ([RFC7540](#) [Section 8.2](#)) extra care must be taken to ensure that the pushed URI

is one that the client would have directed the same query to if the client had initiated the request. This specification does not extend DNS resolution privileges to URIs that are not recognized by the client as trusted DNS API servers.

6.4. Content Negotiation

In order to maximize interoperability, DNS API clients and DNS API servers MUST support the "application/dns-udpwireformat" media type. Other media types MAY be used as defined by HTTP Content Negotiation ([\[RFC7231\] Section 3.4](#)).

7. IANA Considerations

7.1. Registration of application/dns-udpwireformat Media Type

To: ietf-types@iana.org
Subject: Registration of MIME media type
application/dns-udpwireformat

MIME media type name: application

MIME subtype name: dns-udpwireformat

Required parameters: n/a

Optional parameters: original_transport

The "original_transport" parameter has two defined values, "udp" and "tcp". This parameter is only expected to be used by servers.

Encoding considerations: This is a binary format. The contents are a DNS message as defined in [RFC 1035](#). The format used here is for DNS over UDP, which is the format defined in the diagrams in [RFC 1035](#).

Security considerations: The security considerations for carrying this data are the same for carrying DNS without encryption.

Interoperability considerations: None.

Published specification: This document.

Applications that use this media type:

Systems that want to exchange full DNS messages.

Additional information:

Magic number(s): n/a

File extension(s): n/a

Macintosh file type code(s): n/a

Person & email address to contact for further information:

Paul Hoffman, paul.hoffman@icann.org

Intended usage: COMMON

Restrictions on usage: n/a

Author: Paul Hoffman, paul.hoffman@icann.org

Change controller: IESG

8. Security Considerations

Running DNS over HTTPS relies on the security of the underlying HTTP transport. This mitigates classic amplification attacks for UDP-based DNS. Implementations utilizing HTTP/2 benefit from the TLS profile defined in [\[RFC7540\] Section 9.2](#).

Session level encryption has well known weaknesses with respect to traffic analysis which might be particularly acute when dealing with DNS queries. HTTP/2 provides further advice about the use of compression ([Section 10.6 of \[RFC7540\]](#)) and padding ([Section 10.7 of \[RFC7540\]](#)).

The HTTPS connection provides transport security for the interaction between the DNS API server and client, but does not inherently ensure the authenticity of DNS data. A DNS API client may also perform full DNSSEC validation of answers received from a DNS API server or it may choose to trust answers from a particular DNS API server, much as a DNS client might choose to trust answers from its recursive DNS resolver. This capability might be affected by the response media type.

[Section 6.1](#) describes the interaction of this protocol with HTTP caching. An adversary that can control the cache used by the client can affect that client's view of the DNS. This is no different than the security implications of HTTP caching for other protocols that use HTTP.

A server that is acting both as a normal web server and a DNS API server is in a position to choose which DNS names it forces a client to resolve (through its web service) and also be the one to answer those queries (through its DNS API service). An untrusted DNS API server can thus easily cause damage by poisoning a client's cache with names that the DNS API server chooses to poison. A client **MUST NOT** trust a DNS API server simply because it was discovered, or because the client was told to trust the DNS API server by an untrusted party. Instead, a client **MUST** only trust DNS API server that is configured as trustworthy.

A client can use DNS over HTTPS as one of multiple mechanisms to obtain DNS data. If a client of this protocol encounters an HTTP error after sending a DNS query, and then falls back to a different DNS retrieval mechanism, doing so can weaken the privacy and authenticity expected by the user of the client.

9. Operational Considerations

Local policy considerations and similar factors mean different DNS servers may provide different results to the same query: for instance in split DNS configurations [RFC6950]. It logically follows that the server which is queried can influence the end result. Therefore a client's choice of DNS server may affect the responses it gets to its queries. For example, in the case of DNS64 [RFC6147], the choice could affect whether IPv6/IPv4 translation will work at all.

The HTTPS channel used by this specification establishes secure two party communication between the DNS API client and the DNS API server. Filtering or inspection systems that rely on unsecured transport of DNS will not function in a DNS over HTTPS environment.

Some HTTPS client implementations perform real time third party checks of the revocation status of the certificates being used by TLS. If this check is done as part of the DNS API server connection procedure and the check itself requires DNS resolution to connect to the third party a deadlock can occur. The use of OCSP [RFC6960] servers or AIA for CRL fetching ([RFC5280] Section 4.2.2.1) are examples of how this deadlock can happen. To mitigate the possibility of deadlock, DNS API servers SHOULD NOT rely on DNS based references to external resources in the TLS handshake. For OCSP the server can bundle the certificate status as part of the handshake using a mechanism appropriate to the version of TLS, such as using [RFC6066] Section 8 for TLS version 1.2. AIA deadlocks can be avoided by providing intermediate certificates that might otherwise be obtained through additional requests.

A DNS API client may face a similar bootstrapping problem when the HTTP request needs to resolve the hostname portion of the DNS URI. Just as the address of a traditional DNS nameserver cannot be originally determined from that same server, a DNS API client cannot use its DNS API server to initially resolve the server's host name into an address. Alternative strategies a client might employ include making the initial resolution part of the configuration, IP based URIs and corresponding IP based certificates for HTTPS, or resolving the DNS API server's hostname via traditional DNS or another DNS API server while still authenticating the resulting connection via HTTPS.

HTTP [RFC7230] is a stateless application level protocol and therefore DOH implementations do not provide stateful ordering guarantees between different requests. DOH cannot be used as a transport for other protocols that require strict ordering.

If a DNS API server responds to a DNS API client with a DNS message that has the TC (truncation) bit set in the header, that indicates that the DNS API server was not able to retrieve a full answer for the query and is providing the best answer it could get. This protocol does not require that a DNS API server that cannot get an untruncated answer send back such an answer; it can instead send back an HTTP error to indicate that it cannot give a useful answer.

This protocol does not define any use for the "original_transport" optional parameter of the application/dns-udpwireformat media type.

10. Acknowledgments

This work required a high level of cooperation between experts in different technologies. Thank you Ray Bellis, Stephane Bortzmeyer, Manu Bretelle, Tony Finch, Daniel Kahn Gilmor, Olafur Guomundsson, Wes Hardaker, Rory Hewitt, Joe Hildebrand, David Lawrence, Eliot Lear, John Mattson, Alex Mayrhofer, Mark Nottingham, Jim Reid, Adam Roach, Ben Schwartz, Davey Song, Daniel Stenberg, Andrew Sullivan, Martin Thomson, and Sam Weiler.

11. References

11.1. Normative References

- [RFC1035] Mockapetris, P., "Domain names - implementation and specification", STD 13, [RFC 1035](#), DOI 10.17487/RFC1035, November 1987, <<https://www.rfc-editor.org/info/rfc1035>>.
- [RFC2308] Andrews, M., "Negative Caching of DNS Queries (DNS NCACHE)", [RFC 2308](#), DOI 10.17487/RFC2308, March 1998, <<https://www.rfc-editor.org/info/rfc2308>>.
- [RFC4648] Josefsson, S., "The Base16, Base32, and Base64 Data Encodings", [RFC 4648](#), DOI 10.17487/RFC4648, October 2006, <<https://www.rfc-editor.org/info/rfc4648>>.
- [RFC5246] Dierks, T. and E. Rescorla, "The Transport Layer Security (TLS) Protocol Version 1.2", [RFC 5246](#), DOI 10.17487/RFC5246, August 2008, <<https://www.rfc-editor.org/info/rfc5246>>.
- [RFC6570] Gregorio, J., Fielding, R., Hadley, M., Nottingham, M., and D. Orchard, "URI Template", [RFC 6570](#), DOI 10.17487/RFC6570, March 2012, <<https://www.rfc-editor.org/info/rfc6570>>.

- [RFC7230] Fielding, R., Ed. and J. Reschke, Ed., "Hypertext Transfer Protocol (HTTP/1.1): Message Syntax and Routing", [RFC 7230](#), DOI 10.17487/RFC7230, June 2014, <<https://www.rfc-editor.org/info/rfc7230>>.
- [RFC7231] Fielding, R., Ed. and J. Reschke, Ed., "Hypertext Transfer Protocol (HTTP/1.1): Semantics and Content", [RFC 7231](#), DOI 10.17487/RFC7231, June 2014, <<https://www.rfc-editor.org/info/rfc7231>>.
- [RFC7234] Fielding, R., Ed., Nottingham, M., Ed., and J. Reschke, Ed., "Hypertext Transfer Protocol (HTTP/1.1): Caching", [RFC 7234](#), DOI 10.17487/RFC7234, June 2014, <<https://www.rfc-editor.org/info/rfc7234>>.
- [RFC7540] Belshe, M., Peon, R., and M. Thomson, Ed., "Hypertext Transfer Protocol Version 2 (HTTP/2)", [RFC 7540](#), DOI 10.17487/RFC7540, May 2015, <<https://www.rfc-editor.org/info/rfc7540>>.
- [RFC7541] Peon, R. and H. Ruellan, "HPACK: Header Compression for HTTP/2", [RFC 7541](#), DOI 10.17487/RFC7541, May 2015, <<https://www.rfc-editor.org/info/rfc7541>>.
- [RFC8174] Leiba, B., "Ambiguity of Uppercase vs Lowercase in [RFC 2119](#) Key Words", [BCP 14](#), [RFC 8174](#), DOI 10.17487/RFC8174, May 2017, <<https://www.rfc-editor.org/info/rfc8174>>.

11.2. Informative References

- [CORS] "Cross-Origin Resource Sharing", n.d., <<https://fetch.spec.whatwg.org/#http-cors-protocol>>.
- [RFC2181] Elz, R. and R. Bush, "Clarifications to the DNS Specification", [RFC 2181](#), DOI 10.17487/RFC2181, July 1997, <<https://www.rfc-editor.org/info/rfc2181>>.
- [RFC5280] Cooper, D., Santesson, S., Farrell, S., Boeyen, S., Housley, R., and W. Polk, "Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile", [RFC 5280](#), DOI 10.17487/RFC5280, May 2008, <<https://www.rfc-editor.org/info/rfc5280>>.
- [RFC5861] Nottingham, M., "HTTP Cache-Control Extensions for Stale Content", [RFC 5861](#), DOI 10.17487/RFC5861, May 2010, <<https://www.rfc-editor.org/info/rfc5861>>.

- [RFC6066] Eastlake 3rd, D., "Transport Layer Security (TLS) Extensions: Extension Definitions", [RFC 6066](#), DOI 10.17487/RFC6066, January 2011, <<https://www.rfc-editor.org/info/rfc6066>>.
- [RFC6147] Bagnulo, M., Sullivan, A., Matthews, P., and I. van Beijnum, "DNS64: DNS Extensions for Network Address Translation from IPv6 Clients to IPv4 Servers", [RFC 6147](#), DOI 10.17487/RFC6147, April 2011, <<https://www.rfc-editor.org/info/rfc6147>>.
- [RFC6891] Damas, J., Graff, M., and P. Vixie, "Extension Mechanisms for DNS (EDNS(0))", STD 75, [RFC 6891](#), DOI 10.17487/RFC6891, April 2013, <<https://www.rfc-editor.org/info/rfc6891>>.
- [RFC6950] Peterson, J., Kolkman, O., Tschafenig, H., and B. Aboba, "Architectural Considerations on Application Features in the DNS", [RFC 6950](#), DOI 10.17487/RFC6950, October 2013, <<https://www.rfc-editor.org/info/rfc6950>>.
- [RFC6960] Santesson, S., Myers, M., Ankney, R., Malpani, A., Galperin, S., and C. Adams, "X.509 Internet Public Key Infrastructure Online Certificate Status Protocol - OCSP", [RFC 6960](#), DOI 10.17487/RFC6960, June 2013, <<https://www.rfc-editor.org/info/rfc6960>>.
- [RFC7719] Hoffman, P., Sullivan, A., and K. Fujiwara, "DNS Terminology", [RFC 7719](#), DOI 10.17487/RFC7719, December 2015, <<https://www.rfc-editor.org/info/rfc7719>>.
- [RFC7858] Hu, Z., Zhu, L., Heidemann, J., Mankin, A., Wessels, D., and P. Hoffman, "Specification for DNS over Transport Layer Security (TLS)", [RFC 7858](#), DOI 10.17487/RFC7858, May 2016, <<https://www.rfc-editor.org/info/rfc7858>>.

11.3. URIs

- [1] <https://github.com/dohwg/draft-ietf-doh-dns-over-https>

Appendix A. Previous Work on DNS over HTTP or in Other Formats

The following is an incomplete list of earlier work that related to DNS over HTTP/1 or representing DNS data in other formats.

The list includes links to the tools.ietf.org site (because these documents are all expired) and web sites of software.

- o <https://tools.ietf.org/html/draft-mohan-dns-query-xml>
- o <https://tools.ietf.org/html/draft-daley-dnsxml>
- o <https://tools.ietf.org/html/draft-dulaunoy-dnsop-passive-dns-cof>
- o <https://tools.ietf.org/html/draft-bortzmeyer-dns-json>
- o <https://www.nlnetlabs.nl/projects/dnssec-trigger/>

Authors' Addresses

Paul Hoffman
ICANN

Email: paul.hoffman@icann.org

Patrick McManus
Mozilla

Email: mcmanus@ducksong.com

