Network Working Group
Internet-Draft
Intended status: Informational
Expires: January 9, 2020

M. Thomson
Mozilla

July 08, 2019

# Long-term Viability of Protocol Extension Mechanisms
## draft-thomson-use-it-or-lose-it-04

Abstract

The ability to change protocols depends on exercising the extension and version negotiation mechanisms that support change. Protocols that don't use these mechanisms can find that deploying changes can be difficult and costly.

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 January 9, 2020.

Table of Contents



## 1. Introduction

A successful protocol [SUCCESS] will change in ways that allow it to continue to fulfill the needs of its users. New use cases, conditions and constraints on the deployment of a protocol can render a protocol that does not change obsolete.

Usage patterns and requirements for a protocol shift over time. In response, implementations might adjust usage patterns within the constraints of the protocol, the protocol could be extended, or a replacement protocol might be developed. Experience with Internet-scale protocol deployment shows that each option comes with different costs. [TRANSITIONS] examines the problem of protocol evolution more broadly.

This document examines the specific conditions that determine whether protocol maintainers have the ability to design and deploy new or modified protocols. Section 2 highlights some historical issues with difficulties in transitions to new protocol features. Section 3 argues that ossified protocols are more difficult to update and successful protocols make frequent use of new extensions and code-points. Section 4 outlines several strategies that might aid in ensuring that protocol changes remain possible over time.

The experience that informs this document is predominantly at "higher" layers of the network stack, in protocols that operate at

very large scale and Internet-scale applications. It is possible that these conclusions are less applicable to protocol deployments that have less scale and diversity, or operate under different constraints.

## 2. Implementations of Protocols are Imperfect

A change to a protocol can be made extremely difficult to deploy if there are bugs in implementations with which the new deployment needs to interoperate. Bugs in the handling of new codepoints or extensions can mean that instead of handling the mechanism as designed, endpoints react poorly. This can manifest as abrupt termination of sessions, errors, crashes, or disappearances of endpoints and timeouts.

Interoperability with other implementations is usually highly valued, so deploying mechanisms that trigger adverse reactions can be untenable. Where interoperability is a competitive advantage, this is true even if the negative reactions happen infrequently or only under relatively rare conditions.

Deploying a change to a protocol could require implementations fix a substantial proportion of the bugs that the change exposes. This can involve a difficult process that includes identifying the cause of these errors, finding the responsible implementation(s), coordinating a bug fix and release plan, contacting users and/or the operator of affected services, and waiting for the fix to be deployed.

Given the effort involved in fixing problems, the existence of these sorts of bugs can outright prevent the deployment of some types of protocol changes, especially for protocols involving multiple parties or that are considered critical infrastructure (e.g., IP, BGP, DNS, or TLS). It could even be necessary to come up with a new protocol design that uses a different method to achieve the same result.

The set of interoperable features in a protocol is often the subset of its features that have some value to those implementing and deploying the protocol. It is not always the case that future extensibility is in that set.

### 2.1. Good Protocol Design is Not Itself Sufficient

It is often argued that the design of a protocol extension point or version negotiation capability is critical to the freedom that it ultimately offers.

RFC 6709 [EXTENSIBILITY] contains a great deal of well-considered advice on designing for extension. It includes the following advice:

> This means that, to be useful, a protocol version- negotiation mechanism should be simple enough that it can reasonably be assumed that all the implementers of the first protocol version at least managed to implement the version-negotiation mechanism correctly.

This has proven to be insufficient in practice. Many protocols have evidence of imperfect implementation of critical mechanisms of this sort. Mechanisms that aren't used are the ones that fail most often. The same paragraph from RFC 6709 acknowledges the existence of this problem, but does not offer any remedy:

> The nature of protocol version-negotiation mechanisms is that, by definition, they don't get widespread real-world testing until _after_ the base protocol has been deployed for a while, and its deficiencies have become evident.

Indeed, basic interoperability is considered critical early in the deployment of a protocol. Race-to-market attitudes frequently result in an engineering practice that values simplicity will tend to make version negotiation and extension mechanisms optional for this basic interoperability. This leads to these mechanisms being uniquely affected by this problem.

Transport Layer Security (TLS) [TLS12] provides examples of where a design that is objectively sound fails when incorrectly implemented. TLS provides examples of failures in protocol version negotiation and extensibility.

Version negotiation in TLS 1.2 and earlier uses the "Highest mutually supported version (HMSV)" scheme exactly as it is described in [EXTENSIBILITY]. However, clients are unable to advertise a new version without causing a non-trivial proportions of sessions to fail due to bugs in server and middlebox implementations.

Intolerance to new TLS versions is so severe [INTOLERANCE] that TLS 1.3 [TLS13] has abandoned HMSV version negotiation for a new mechanism.

The server name indication (SNI) [TLS-EXT] in TLS is another excellent example of the failure of a well-designed extensibility point. SNI uses the same technique for extension that is used with considerable success in other parts of the TLS protocol. The original design of SNI includes the ability to include multiple names of different types.

What is telling in this case is that SNI was defined with just one type of name: a domain name. No other type has ever been

standardized, though several have been proposed. Despite an otherwise exemplary design, SNI is so inconsistently implemented that any hope for using the extension point it defines has been abandoned [SNI].

Requiring simplistic processing steps when encountering unknown conditions, such as unsupported version numbers, can potentially prevent these sorts of situations. A counter example is the first version of the Simple Network Management Protocol (SNMP), where an unparseable and an authentication message are treated the same way by the server: no response is generated [SNMPv1]:

> It then verifies the version number of the SNMP message. If there is a mismatch, it discards the datagram and performs no further actions.

When SNMP versions 2, 2c and 3 came along, older agents did exactly what the protocol specifies should have done: dropped it from being processing without returning a response. This was likely successful because there was no requirement to create and return an elaborate error response to the client.

## 2.2. Multi-Party Interactions and Middleboxes

Even the most superficially simple protocols can often involve more actors than is immediately apparent. A two-party protocol has two ends, but even at the endpoints of an interaction, protocol elements can be passed on to other entities in ways that can affect protocol operation.

One of the key challenges in deploying new features is ensuring compatibility with all actors that could be involved in the protocol.

Protocols deployed without active measures against intermediation will tend to become intermediated over time, as network operators deploy middleboxes to perform some function on traffic [PATH-SIGNALS]. In particular, one of the consequences of an unencrypted protocol is that any element on path can interact with the protocol. For example, HTTP was specifically designed with intermediation in mind, transparent proxies [HTTP] are not only possible but sometimes advantageous, despite some significant downsides. Consequently, transparent proxies for cleartext HTTP are commonplace. The DNS protocol was designed with intermediation in mind through its use of caching recursive resolvers [DNS]. What was less anticipated was the forced spoofing of DNS records by many middle-boxes such as those that inject authentication or pay-wall mechanisms as an authentication and authorization check, which are now prevalent in hotels, coffee shops and business networks.

Middleboxes are also protocol participants, to the degree that they are able to observe and act in ways that affect the protocol. The degree to which a middlebox participates varies from the basic functions that a router performs to full participation. For example, a SIP back-to-back user agent (B2BUA) [B2BUA] can be very deeply involved in the SIP protocol.

This phenomenon appears at all layers of the protocol stack, even when protocols are not designed with middlebox participation in mind. TCP's [TCP] extension points have been rendered difficult to use, largely due to middlebox interactions, as experience with Multipath TCP [MPTCP] and Fast Open [TFO] has shown. IP's version field was rendered useless when encapsulated over Ethernet, requring a new ethertype with IPv6 [RFC2462], due in part to layer 2 devices making version-independent assumptions about the structure of the IPv4 header.

By increasing the number of different actors involved in any single protocol exchange, the number of potential implementation bugs that a deployment needs to contend with also increases. In particular, incompatible changes to a protocol that might be negotiated between endpoints in ignorance of the presence of a middlebox can result in a middlebox interfering in negative and unexpected ways.

Unfortunately, middleboxes can considerably increase the difficulty of deploying new versions or other changes to a protocol.

## 3. Retaining Viable Protocol Evolution Mechanisms

The design of a protocol for extensibility and eventual replacement [EXTENSIBILITY] does not guarantee the ability to exercise those options. The set of features that enable future evolution need to be interoperable in the first implementations and deployments of the protocol. Active use of mechanisms that support evolution is the only way to ensure that they remain available for new uses.

The conditions for retaining the ability to evolve a design is most clearly evident in the protocols that are known to have viable version negotiation or extension points. The definition of mechanisms alone is insufficient; it's the active use of those mechanisms that determines the existence of freedom. Protocols that routinely add new extensions and code points rarely have trouble adding additional ones, especially when unknown code-points and extensions are to be safely ignored when not understood.

### 3.1. Examples of Active Use

For example, header fields in email [SMTP], HTTP [HTTP] and SIP [SIP] all derive from the same basic design, which amounts to a list name/value pairs. There is no evidence of significant barriers to deploying header fields with new names and semantics in email and HTTP as clients and servers can ignore headers they do not understand or need. The widespread deployment of SIP B2BUAs means that new SIP header fields do not reliably reach peers, however, which doesn't necessarily cause interoperability issues but rather does cause feature deployment issues.

In another example, the attribute-value pairs (AVPs) in Diameter [DIAMETER] are fundamental to the design of the protocol. Any use of Diameter requires exercising the ability to add new AVPs. This is routinely done without fear that the new feature might not be successfully deployed.

Ossified DNS code bases and systems resulted in fears that new Resource Record Codes (RRCodes) would take years of software propagation before new RRCodes could be used. The result for a long time was heavily overloaded use of the TXT record, such as in the Sender Policy Framework [SPF]. It wasn't until after the standard mechanism for dealing with new RRCodes [RRTYPE] was considered widely deployed that new RRCodes can be safely created and used immediately.

These examples show extension points that are heavily used are also being relatively unaffected by deployment issues preventing addition of new values for new use cases.

These examples also confirm the case that good design is not a prerequisite for success. On the contrary, success is often despite shortcomings in the design. For instance, the shortcomings of HTTP header fields are significant enough that there are ongoing efforts to improve the syntax [HTTP-HEADERS].

Only by using a protocol's extension capabilities does it ensure the availability of that capability. Protocols that fail to use a mechanism, or a protocol that only rarely uses a mechanism, may suffer an inability to rely on that mechanism.

### 3.2. Dependency is Better

The best way to guarantee that a protocol mechanism is used is to make it critical to an endpoint participating in that protocol. This means that implementations rely on both the existence of the protocol mechanism and its use.

For example, the message format in SMTP relies on header fields for most of its functions, including the most basic functions. A deployment of SMTP cannot avoid including an implementation of header field handling. In addition to this, the regularity with which new header fields are defined and used ensures that deployments frequently encounter header fields that it does not understand. An SMTP implementation therefore needs to be able to both process header fields that it understands and ignore those that it does not.

In this way, implementing the extensibility mechanism is not merely mandated by the specification, it is crucial to the functioning of a protocol deployment. Should an implementation fail to correctly implement the mechanism, that failure would quickly become apparent.

Caution is advised to avoid assuming that building a dependency on an extension mechanism is sufficient to ensure availability of that mechanism in the long term. If the set of possible uses is narrowly constrained and deployments do not change over time, implementations might not see new variations or assume a narrower interpretation of what is possible. Those implementations might still exhibit errors when presented with a new variation.

### 3.3. Unused Extension Points Become Unusable

In contrast, there are many examples of extension points in protocols that have been either completely unused, or their use was so infrequent that they could no longer be relied upon to function correctly.

HTTP has a number of very effective extension points in addition to the aforementioned header fields. It also has some examples of extension point that are so rarely used that it is possible that they are not at all usable. Extension points in HTTP that might be unwise to use include the extension point on each chunk in the chunked transfer coding [HTTP], the ability to use transfer codings other than the chunked coding, and the range unit in a range request [HTTP-RANGE].

Even where extension points have multiple valid values, if the set of permitted values does not change over time, there is still a risk that new values are not tolerated by existing implementations. If the set of values for a particular field remains fixed over a long period, some implementations might not correctly handle a new value when it is introduced. For example, implementations of TLS broke when new values of the signature_algorithms extension were introduced.

Codepoints that are reserved for future use can be especially problematic. Reserving codepoints without attributing semantics to their use can result in diverse or conflicting semantics being attributed without any hope of interoperability. An example of this is the "class E" address space in IPv4 [RFC0988], which was reserved without assigning any semantics. For protocols that can use negotiation to attribute semantics to codepoints, it is possible that unused codepoints can be reclaimed for active use, though this requires that the negotiation include all protocol participants.

## 4. Defensive Design Principles for Protocols

There are several potential approaches that can provide some measure of protection against a protocol deployment becoming resistant to change.

### 4.1. Active Use

As discussed in Section 3, the most effective defense against misuse of protocol extension points is active use.

Implementations are most likely to be tolerant of new values if they depend on being able to use new values. Failing that, implementations that routinely see new values are more likely to correctly handle new values. More frequent changes will improve the likelihood that incorrect handling or intolerance is discovered and rectified. The longer an intolerant implementation is deployed, the more difficult it is to correct.

What active use means could depend greatly on the environment in which a protocol is deployed. The frequency of changes necessary to safeguard some mechanisms might be slow enough to attract ossification in another protocol deployment, while being excessive in others. There are currently no firm guidelines for new protocol development, as much is being learned about what techniques are most effective.

### 4.2. Cryptography

Cryptography can be used to reduce the number of entities that can participate in a protocol. Using tools like TLS ensures that only authorized participants are able to influence whether a new protocol feature is used.

Permitting fewer protocol participants reduces the number of implementations that can prevent a new mechanism from being deployed. As recommended in [PATH-SIGNALS], use of encryption and integrity protection can be used to limit participation.

For example, the QUIC protocol [QUIC] adopts both encryption and integrity protection. Encryption is used to carefully control what information is exposed to middleboxes. For those fields that are not encrypted, QUIC uses integrity protection to prevent modification.

## 4.3. Grease

"Grease" [GREASE] identifies lack of use as an issue (protocol mechanisms "rusting" shut) and proposes reserving values for extensions that have no semantic value attached.

The design in [GREASE] is aimed at the style of negotiation most used in TLS, where the client offers a set of options and the server chooses the one that it most prefers from those that it supports. A client that uses grease randomly offers options - usually just one - from a set of reserved values. These values are guaranteed to never be assigned real meaning, so the server will never have cause to genuinely select one of these values.

More generally, greasing is used to refer to any attempt to exercise extension points without changing endpoint behavior, other than to encourage participants to tolerate new or varying values of protocol elements.

The principle that grease operates on is that an implementation that is regularly exposed to unknown values is less likely to be intolerant of new values when they appear. This depends largely on the assumption that the difficulty of implementing the extension mechanism correctly is not significantly more effort than implementing code to identify and filter out reserved values. Reserving random or unevenly distributed values for this purpose is thought to further discourage special treatment.

Without reserved greasing codepoints, an implementation can use code points from spaces used for private or experimental use if such a range exists. In addition to the risk of triggering participation in an unwanted experiment, this can be less effective. Incorrect implementations might still be able to correctly identify these code points and ignore them.

In addition to advertising bogus capabilities, an endpoint might also selectively disable non-critical protocol elements to test the ability of peers to handle the absence of certain capabilities.

This style of defensive design is limited because it is only superficial. It only exercises a small part of the mechanisms that support extensibility. More critically, it does not easily translate to all forms of extension point. For instance, HMSV negotiation

cannot be greased in this fashion. Other techniques might be necessary for protocols that don't rely on the particular style of exchange that is predominant in TLS.

Grease is deployed with the intent of quickly revealing errors in implementing the mechanisms it safeguards. Though it has been effective at revealing problems in some cases with TLS, the efficacy of greasing isn't proven more generally. Where implementations are able to tolerate a non-zero error rate in their operation, greasing offers a potential option for safeguarding future extensibility.

### 4.4. Invariants

Documenting aspects of the protocol that cannot or will not change as extensions or new versions are added can be a useful exercise. Understanding what aspects of a protocol are invariant can help guide the process of identifying those parts of the protocol that might change.

As a means of protecting extensibility, a declaration of protocol invariants is useful only to the extent that protocol participants are willing to allow new uses for the protocol. Like greasing, protocol participants could still purposefully block the deployment of new features. A protocol that declares protocol invariants relies on implementations understanding and respecting those invariants.

Protocol invariants need to be clearly and concisely documented. Including examples of aspects of the protocol that are not invariant, such as the appendix of [QUIC-INVARIANTS], can be used to clarify intent.

### 4.5. Effective Feedback

While not a direct means of protecting extensibility mechanisms, feedback systems can be important to discovering problems.

Visibility of errors is critical to the success of the grease technique (see Section 4.3). The grease design is most effective if a deployment has a means of detecting and reporting errors. Ignoring errors could allow problems to become entrenched.

Feedback on errors is more important during the development and early deployment of a change. It might also be helpful to disable automatic error recovery methods during development.

Automated feedback systems are important for automated systems, or where error recovery is also automated. For instance, connection failures with HTTP alternative services [ALT-SVC] are not permitted

to affect the outcome of transactions. An automated feedback system for capturing failures in alternative services is therefore necessary for failures to be detected.

How errors are gathered and reported will depend greatly on the nature of the protocol deployment and the entity that receives the report. For instance, end users, developers, and network operations each have different requirements for how error reports are created, managed, and acted upon.

## 5. Security Considerations

The ability to design, implement, and deploy new protocol mechanisms can be critical to security. In particular, it is important to be able to replace cryptographic algorithms over time [AGILITY]. For example, preparing for replacement of weak hash algorithms was made more difficult through misuse [HASH].

## 6. IANA Considerations

This document makes no request of IANA.

## 7. Informative References

[AGILITY] Housley, R., "Guidelines for Cryptographic Algorithm Agility and Selecting Mandatory-to-Implement Algorithms", BCP 201, RFC 7696, DOI 10.17487/RFC7696, November 2015, <https://www.rfc-editor.org/info/rfc7696>.

[ALT-SVC] Nottingham, M., McManus, P., and J. Reschke, "HTTP Alternative Services", RFC 7838, DOI 10.17487/RFC7838, April 2016, <https://www.rfc-editor.org/info/rfc7838>.

[B2BUA] Kaplan, H. and V. Pascual, "A Taxonomy of Session Initiation Protocol (SIP) Back-to-Back User Agents", RFC 7092, DOI 10.17487/RFC7092, December 2013, <https://www.rfc-editor.org/info/rfc7092>.

[DIAMETER] Fajardo, V., Ed., Arkko, J., Loughney, J., and G. Zorn, Ed., "Diameter Base Protocol", RFC 6733, DOI 10.17487/RFC6733, October 2012, <https://www.rfc-editor.org/info/rfc6733>.

[DNS] Mockapetris, P., "Domain names - concepts and facilities", STD 13, RFC 1034, DOI 10.17487/RFC1034, November 1987, <https://www.rfc-editor.org/info/rfc1034>.

[EXTENSIBILITY]
Carpenter, B., Aboba, B., Ed., and S. Cheshire, "Design Considerations for Protocol Extensions", RFC 6709, DOI 10.17487/RFC6709, September 2012, <https://www.rfc-editor.org/info/rfc6709>.

[GREASE] Benjamin, D., "Applying GREASE to TLS Extensibility", draft-ietf-tls-grease-02 (work in progress), January 2019.

[HASH] Bellovin, S. and E. Rescorla, "Deploying a New Hash Algorithm", Proceedings of NDSS '06 , 2006, <https://www.cs.columbia.edu/~smb/papers/new-hash.pdf>.

[HTTP] 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>.

[HTTP-HEADERS]
Nottingham, M. and P. Kamp, "Structured Headers for HTTP", draft-ietf-httpbis-header-structure-10 (work in progress), April 2019.

[HTTP-RANGE]
Fielding, R., Ed., Lafon, Y., Ed., and J. Reschke, Ed., "Hypertext Transfer Protocol (HTTP/1.1): Range Requests", RFC 7233, DOI 10.17487/RFC7233, June 2014, <https://www.rfc-editor.org/info/rfc7233>.

[INTOLERANCE]
Kario, H., "Re: [TLS] Thoughts on Version Intolerance", July 2016, <https://mailarchive.ietf.org/arch/msg/tls/bOJ2JQc3HjAHFFWCiNTIb0JuMZc>.

[MPTCP] Ford, A., Raiciu, C., Handley, M., and O. Bonaventure, "TCP Extensions for Multipath Operation with Multiple Addresses", RFC 6824, DOI 10.17487/RFC6824, January 2013, <https://www.rfc-editor.org/info/rfc6824>.

[PATH-SIGNALS]
Hardie, T., "Transport Protocol Path Signals", draft-iab-path-signals-03 (work in progress), January 2019.

[QUIC] Iyengar, J. and M. Thomson, "QUIC: A UDP-Based Multiplexed and Secure Transport", draft-ietf-quic-transport-20 (work in progress), April 2019.

[QUIC-INVARIANTS]
Thomson, M., "Version-Independent Properties of QUIC", draft-ietf-quic-invariants-04 (work in progress), April 2019.

[RFC0988] Deering, S., "Host extensions for IP multicasting", RFC 988, DOI 10.17487/RFC0988, July 1986, <https://www.rfc-editor.org/info/rfc988>.

[RFC2462] Thomson, S. and T. Narten, "IPv6 Stateless Address Autoconfiguration", RFC 2462, DOI 10.17487/RFC2462, December 1998, <https://www.rfc-editor.org/info/rfc2462>.

[RRTYPE] Gustafsson, A., "Handling of Unknown DNS Resource Record (RR) Types", RFC 3597, DOI 10.17487/RFC3597, September 2003, <https://www.rfc-editor.org/info/rfc3597>.

[SIP] Rosenberg, J., Schulzrinne, H., Camarillo, G., Johnston, A., Peterson, J., Sparks, R., Handley, M., and E. Schooler, "SIP: Session Initiation Protocol", RFC 3261, DOI 10.17487/RFC3261, June 2002, <https://www.rfc-editor.org/info/rfc3261>.

[SMTP] Resnick, P., Ed., "Internet Message Format", RFC 5322, DOI 10.17487/RFC5322, October 2008, <https://www.rfc-editor.org/info/rfc5322>.

[SNI] Langley, A., "Accepting that other SNI name types will never work", March 2016, <https://mailarchive.ietf.org/arch/msg/tls/1t79gzNItZd71DwwoaqcQQ_4Yxc>.

[SNMPv1] Case, J., Fedor, M., Schoffstall, M., and J. Davin, "Simple Network Management Protocol (SNMP)", RFC 1157, DOI 10.17487/RFC1157, May 1990, <https://www.rfc-editor.org/info/rfc1157>.

[SPF] Kitterman, S., "Sender Policy Framework (SPF) for Authorizing Use of Domains in Email, Version 1", RFC 7208, DOI 10.17487/RFC7208, April 2014, <https://www.rfc-editor.org/info/rfc7208>.

[SUCCESS] Thaler, D. and B. Aboba, "What Makes for a Successful Protocol?", RFC 5218, DOI 10.17487/RFC5218, July 2008, <https://www.rfc-editor.org/info/rfc5218>.

[TCP] Postel, J., "Transmission Control Protocol", STD 7, RFC 793, DOI 10.17487/RFC0793, September 1981, <https://www.rfc-editor.org/info/rfc793>.

[TFO] Cheng, Y., Chu, J., Radhakrishnan, S., and A. Jain, "TCP Fast Open", RFC 7413, DOI 10.17487/RFC7413, December 2014, <https://www.rfc-editor.org/info/rfc7413>.

[TLS-EXT] 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>.

[TLS12] 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>.

[TLS13] Rescorla, E., "The Transport Layer Security (TLS) Protocol Version 1.3", RFC 8446, DOI 10.17487/RFC8446, August 2018, <https://www.rfc-editor.org/info/rfc8446>.

[TRANSITIONS] Thaler, D., Ed., "Planning for Protocol Adoption and Subsequent Transitions", RFC 8170, DOI 10.17487/RFC8170, May 2017, <https://www.rfc-editor.org/info/rfc8170>.

## Acknowledgments

Mirja Kuehlewind, Mark Nottingham, and Brian Trammell made significant contributions to this document.

## Author's Address

Martin Thomson
Mozilla

Email: mt@lowentropy.net