

Workgroup: EDM
Internet-Draft:
draft-iab-protocol-maintenance-07
Published: 1 June 2022
Intended Status: Informational
Expires: 3 December 2022
Authors: M. Thomson D. Schinazi
 Mozilla Google LLC

The Harmful Consequences of the Robustness Principle

Abstract

The robustness principle, often phrased as "be conservative in what you send, and liberal in what you accept", has long guided the design and implementation of Internet protocols. The posture this statement advocates promotes interoperability in the short term, but can negatively affect the protocol ecosystem over time. For a protocol that is actively maintained, the robustness principle can, and should, be avoided.

About This Document

This note is to be removed before publishing as an RFC.

The latest revision of this draft can be found at <https://intarchboard.github.io/draft-protocol-maintenance/draft-iab-protocol-maintenance.html>. Status information for this document may be found at <https://datatracker.ietf.org/doc/draft-iab-protocol-maintenance/>.

Discussion of this document takes place on the EDM IAB Program mailing list (<mailto:edm@iab.org>), which is archived at <https://www.iab.org/mailman/listinfo/edm>.

Source for this draft and an issue tracker can be found at <https://github.com/intarchboard/draft-protocol-maintenance>.

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 3 December 2022.

Copyright Notice

Copyright (c) 2022 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.

Table of Contents

- [1. Introduction](#)
- [2. Fallibility of Specifications](#)
- [3. Protocol Decay](#)
- [4. Ecosystem Effects](#)
- [5. Active Protocol Maintenance](#)
- [6. Extensibility](#)
- [7. Virtuous Intolerance](#)
- [8. Exclusion](#)
- [9. Security Considerations](#)
- [10. IANA Considerations](#)
- [11. Informative References](#)
- [Acknowledgments](#)
- [Authors' Addresses](#)

1. Introduction

The robustness principle has been hugely influential in shaping the design of the Internet. As stated in the IAB document on Architectural Principles of the Internet [[RFC1958](#)], the robustness principle advises to:

Be strict when sending and tolerant when receiving.
Implementations must follow specifications precisely when sending to the network, and tolerate faulty input from the network. When in doubt, discard faulty input silently, without returning an error message unless this is required by the specification.

This simple statement captures a significant concept in the design of interoperable systems. Many consider the application of the robustness principle to be instrumental in the success of the

Internet as well as the design of interoperable protocols in general.

Time and experience shows that negative consequences to interoperability accumulate over time if implementations apply the robustness principle. This problem originates from an assumption implicit in the principle that it is not possible to effect change in a system the size of the Internet. It assumes that once a protocol specification is published, changes that might require existing implementations to change are not feasible.

Many problems that the robustness principle was intended to solve can instead be better addressed by active maintenance. Active protocol maintenance is where a community of protocol designers, implementers, and deployers work together to continuously improve and evolve protocol specifications alongside implementations and deployments of those protocols. A community that takes an active role in the maintenance of protocols will no longer need to rely on the robustness principle to avoid interoperability issues.

There is good evidence to suggest that many important protocols are routinely maintained beyond their inception. In particular, a sizeable proportion of IETF activity is dedicated to the stewardship of existing protocols. This document serves primarily as a record of the hazards in applying the robustness principle too broadly, and offers an alternative strategy for handling interoperability problems in deployments.

Ideally, protocol implementations can be actively maintained and never have to apply the robustness principle. The robustness principle may still need to be used as a short-term mitigation for deployments that cannot yet be easily updated and do not yet have documented specifications for workarounds, but such cases need not be permanent. This is discussed further in [Section 5](#).

Avoiding use of the robustness principle does not mean that implementations will be inflexible to new input. As discussed in [Section 6](#), the flexibility to handle future extensions is better supported by specifications being very clear about their extension mechanisms, and implementations being strict in following the requirements of those specifications.

2. Fallibility of Specifications

The context from which the robustness principle was developed provides valuable insights into its intent and purpose. The earliest form of the principle in the RFC series (the Internet Protocol

specification [[RFC0760](#)]) is preceded by a sentence that reveals the motivation for the principle:

While the goal of this specification is to be explicit about the protocol there is the possibility of differing interpretations. In general, an implementation should be conservative in its sending behavior, and liberal in its receiving behavior.

This formulation of the principle expressly recognizes the possibility that the specification could be imperfect. This contextualizes the principle in an important way.

Imperfect specifications are unavoidable, largely because it is more important to proceed to implementation and deployment than it is to perfect a specification. A protocol benefits greatly from experience with its use. A deployed protocol is immeasurably more useful than a perfect protocol specification. This is particularly true in early phases of system design, to which the robustness principle is best suited.

As demonstrated by the IAB document on Successful Protocols [[RFC5218](#)], success or failure of a protocol depends far more on factors like usefulness than on technical excellence. Timely publication of protocol specifications, even with the potential for flaws, likely contributed significantly to the eventual success of the Internet.

This premise that specifications will be imperfect is correct -- however, the robustness principle is almost always the incorrect solution to the problem.

3. Protocol Decay

The application of the robustness principle to any system that is in early phases of deployment, such as the early Internet, is expedient. Applying the principle defers the effort of dealing with interoperability problems and prioritizes progress. However, this deferral can amplify the ultimate cost of handling interoperability problems.

Divergent implementations of a specification emerge over time. When variations occur in the interpretation or expression of semantic components, implementations cease to be perfectly interoperable.

Implementation bugs are often identified as the cause of variation, though it is often a combination of factors. Using a protocol in ways that were not anticipated in the original design, or ambiguities and errors in the specification are often contributing factors. Disagreements on the interpretation of specifications should be expected over the lifetime of a protocol.

Even with the best intentions to maintain protocol correctness, the pressure to interoperate can be significant. No implementation can hope to avoid having to trade correctness for interoperability indefinitely.

An implementation that reacts to variations in the manner recommended in the robustness principle enters a pathological feedback cycle. Over time:

- *Implementations progressively add logic to constrain how data is transmitted, or to permit variations in what is received.

- *Errors in implementations or confusion about semantics are permitted or ignored.

- *These errors can become entrenched, forcing other implementations to be tolerant of those errors.

A flaw can become entrenched as a de facto standard. Any implementation of the protocol is required to replicate the aberrant behavior, or it is not interoperable. This is both a consequence of applying the robustness principle, and a product of a natural reluctance to avoid fatal error conditions. Ensuring interoperability in this environment is often referred to as aiming to be "bug for bug compatible".

For example, in TLS [[TLS](#)], extensions use a tag-length-value format and can be added to messages in any order. However, some server implementations terminated connections if they encountered a TLS ClientHello message that ends with an empty extension. To maintain interoperability, client implementations were required to be aware of this bug and ensure that a ClientHello message ends in a non-empty extension.

The original JSON specification [[RFC4627](#)] demonstrates the effect of specification shortcomings: it did not tightly specify some important details including Unicode handling, ordering and duplication of object members, and number encoding. Consequently, a range of interpretations were used by implementations. An updated JSON specification [[RFC7159](#)] did not correct these errors, concentrating instead on identifying the interoperable subset of JSON. I-JSON [[RFC7493](#)] takes that subset and defines a new format that prohibits the problematic parts of JSON. Of course, that means that I-JSON is not fully interoperable with JSON. Consequently, I-JSON is not widely implemented in parsers. Many JSON parsers now implement the more precise algorithm specified in [[ECMA262](#)].

The robustness principle therefore encourages a chain reaction that can create interoperability problems over time. In particular, the application of the robustness principle is particularly deleterious

for early implementations of new protocols as quirks in early implementations can affect all subsequent deployments.

4. Ecosystem Effects

From observing widely deployed protocols, it appears there are two stable points on the spectrum between being strict versus permissive in the presence of protocol errors:

*If implementations predominantly enforce strict compliance with specifications, newer implementations will experience failures if they do not comply with protocol requirements. Newer implementations need to fix compliance issues in order to be successfully deployed. This ensures that most deployments are compliant.

*Conversely, if non-compliance is tolerated by existing implementations, non-compliant implementations can be deployed successfully. Newer implementations then have strong incentive to tolerate any existing non-compliance in order to be successfully deployed. This ensures that most deployments are tolerant of the same non-compliant behavior.

This happens because interoperability requirements for protocol implementations are set by other deployments. Specifications and - where they exist - conformance test suites might guide the initial development of implementations, but implementations ultimately need to interoperate with deployed implementations.

For widely used protocols, the massive scale of the Internet makes large-scale interoperability testing infeasible for all but a privileged few. The cost of building a new implementation using reverse engineering increases as the number of implementations and bugs increases. Worse, the set of tweaks necessary for wide interoperability can be difficult to discover. In the worst case, a new implementer might have to choose between deployments that have diverged so far as to no longer be interoperable.

Consequently, new implementations might be forced into niche uses, where the problems arising from interoperability issues can be more closely managed. However, restricting new implementations into limited deployments risks causing forks in the protocol. If implementations do not interoperate, little prevents those implementations from diverging more over time.

This has a negative impact on the ecosystem of a protocol. New implementations are key to the continued viability of a protocol. New protocol implementations are also more likely to be developed for new and diverse use cases and are often the origin of features and capabilities that can be of benefit to existing users.

The need to work around interoperability problems also reduces the ability of established implementations to change. An accumulation of mitigations for interoperability issues makes implementations more difficult to maintain and can constrain extensibility (see also the IAB document on the Long-Term Viability of Protocol Extension Mechanisms [[RFC9170](#)]).

Sometimes what appear to be interoperability problems are symptomatic of issues in protocol design. A community that is willing to make changes to the protocol, by revising or extending it, makes the protocol better in the process. Applying the robustness principle instead conceals problems, making it harder, or even impossible, to fix them later.

5. Active Protocol Maintenance

The robustness principle can be highly effective in safeguarding against flaws in the implementation of a protocol by peers. Especially when a specification remains unchanged for an extended period of time, incentive to be tolerant of errors accumulates over time. Indeed, when faced with divergent interpretations of an immutable specification, the only way for an implementation to remain interoperable is to be tolerant of differences in interpretation and implementation errors.

From this perspective, application of the robustness principle to the implementation of a protocol specification that does not change is logical, even necessary. But that conclusion relies on an assumption that existing specifications and implementations cannot change. Applying the robustness principle in this way disproportionately values short-term gains over the negative effects on future implementations and the protocol as a whole.

For a protocol to have sustained viability, it is necessary for both specifications and implementations to be responsive to changes, in addition to handling new and old problems that might arise over time.

Maintaining specifications so that they closely match deployments ensures that implementations are consistently interoperable and removes needless barriers for new implementations. Maintenance also enables continued improvement of the protocol. New use cases are an indicator that the protocol could be successful [[RFC5218](#)].

Protocol designers are strongly encouraged to continue to maintain and evolve protocol specifications beyond their initial inception and definition. This might require the development of revised specifications, extensions, or other supporting material that documents the current state of the protocol. Involvement of those

who implement and deploy the protocol is a critical part of this process, as they provide input on their experience with how the protocol is used.

Most interoperability problems do not require revision of protocols or protocol specifications. For instance, the most effective means of dealing with a defective implementation in a peer could be to contact the developer responsible. It is far more efficient in the long term to fix one isolated bug than it is to deal with the consequences of workarounds.

Early implementations of protocols have a stronger obligation to closely follow specifications as their behavior will affect all subsequent implementations. In addition to specifications, later implementations will be guided by what existing deployments accept. Tolerance of errors in early deployments is most likely to result in problems. Protocol specifications might need more frequent revision during early deployments to capture feedback from early rounds of deployment.

Neglect can quickly produce the negative consequences this document describes. Restoring the protocol to a state where it can be maintained involves first discovering the properties of the protocol as it is deployed, rather than the protocol as it was originally documented. This can be difficult and time-consuming, particularly if the protocol has a diverse set of implementations. Such a process was undertaken for HTTP [[HTTP](#)] after a period of minimal maintenance. Restoring HTTP specifications to relevance took significant effort.

Maintenance is most effective if it is responsive, which is greatly affected by how rapidly protocol changes can be deployed. For protocol deployments that operate on longer time scales, temporary workarounds following the spirit of the robustness principle might be necessary. For this, improvements in software update mechanisms ensure that the cost of reacting to changes is much lower than it was in the past. Alternatively, if specifications can be updated more readily than deployments, details of the workaround can be documented, including the desired form of the protocols once the need for workarounds no longer exists and plans for removing the workaround.

6. Extensibility

Good extensibility [[EXT](#)] can make it easier to respond to new use cases or changes in the environment in which the protocol is deployed.

The ability to extend a protocol is sometimes mistaken for an application of the robustness principle. After all, if one party wants to start using a new feature before another party is prepared to receive it, it might be assumed that the receiving party is being tolerant of unexpected inputs.

A well-designed extensibility mechanism establishes clear rules for the handling of things like new messages or parameters. This depends on precisely specifying the handling of malformed or illegal inputs so that implementations behave consistently in all cases that might affect interoperation. If extension mechanisms and error handling are designed and implemented correctly, new protocol features can be deployed with confidence in the understanding of the effect they have on existing implementations.

In contrast, relying on implementations to consistently apply the robustness principle is not a good strategy for extensibility. Using undocumented or accidental features of a protocol as the basis of an extensibility mechanism can be extremely difficult, as is demonstrated by the case study in [Appendix A.3](#) of [\[EXT\]](#).

A protocol could be designed to permit a narrow set of valid inputs, or it could allow a wide range of inputs as a core feature (see for example [\[HTML\]](#)). Specifying and implementing a more flexible protocol is more difficult; allowing less variability is preferable in the absence of strong reasons to be flexible.

7. Virtuous Intolerance

A well-specified protocol includes rules for consistent handling of aberrant conditions. This increases the chances that implementations will have consistent and interoperable handling of unusual conditions.

Choosing to generate fatal errors for unspecified conditions instead of attempting error recovery can ensure that faults receive attention. This intolerance can be harnessed to reduce occurrences of aberrant implementations.

Intolerance toward violations of specification improves feedback for new implementations in particular. When a new implementation encounters a peer that is intolerant of an error, it receives strong feedback that allows the problem to be discovered quickly.

To be effective, intolerant implementations need to be sufficiently widely deployed that they are encountered by new implementations with high probability. This could depend on multiple implementations deploying strict checks.

This does not mean that intolerance of errors in early deployments of protocols has the effect of preventing interoperability. On the contrary, when existing implementations follow clearly-specified error handling, new implementations or features can be introduced more readily as the effect on existing implementations can be easily predicted; see also [Section 6](#).

Any intolerance also needs to be strongly supported by specifications, otherwise they encourage fracturing of the protocol community or proliferation of workarounds; see [Section 8](#).

Intolerance can be used to motivate compliance with any protocol requirement. For instance, the INADEQUATE_SECURITY error code and associated requirements in HTTP/2 [[HTTP/2](#)] resulted in improvements in the security of the deployed base.

8. Exclusion

Any protocol participant that is affected by changes arising from maintenance might be excluded if they are unwilling or unable to implement or deploy changes that are made to the protocol.

Deliberate exclusion of problematic implementations is an important tool that can ensure that the interoperability of a protocol remains viable. While compatible changes are always preferable to incompatible ones, it is not always possible to produce a design that protects the ability of all current and future protocol participants to interoperate. Developing and deploying changes that risk exclusion of previously interoperating implementations requires some care, but changes to a protocol should not be blocked on the grounds of the risk of exclusion alone.

Exclusion is a direct goal when choosing to be intolerant of errors (see [Section 7](#)). Exclusionary actions are employed with the deliberate intent of protecting future interoperability.

Excluding implementations or deployments can lead to a fracturing of the protocol system that could be more harmful than any divergence resulting from following the robustness principle. The IAB document on Uncoordinated Protocol Development Considered Harmful [[RFC5704](#)] describes how conflict or competition in the maintenance of protocols can lead to similar problems.

9. Security Considerations

Sloppy implementations, lax interpretations of specifications, and uncoordinated extrapolation of requirements to cover gaps in specification can result in security problems. Hiding the consequences of protocol variations encourages the hiding of issues, which can conceal bugs and make them difficult to discover.

The consequences of the problems described in this document are especially acute for any protocol where security depends on agreement about semantics of protocol elements. For instance, use of unsafe security mechanisms, such as weak primitives [MD5] or obsolete mechanisms [SSL3], are good examples of where forcing exclusion (Section 8) can be desirable.

10. IANA Considerations

This document has no IANA actions.

11. Informative References

- [ECMA262] "ECMAScript(R) 2018 Language Specification", ECMA-262 9th Edition, June 2018, <<https://www.ecma-international.org/publications/standards/Ecma-262.htm>>.
- [EXT] 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/rfc/rfc6709>>.
- [HTML] "HTML", WHATWG Living Standard, 8 March 2019, <<https://html.spec.whatwg.org/>>.
- [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/rfc/rfc7230>>.
- [HTTP/2] 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/rfc/rfc7540>>.
- [MD5] Turner, S. and L. Chen, "Updated Security Considerations for the MD5 Message-Digest and the HMAC-MD5 Algorithms", RFC 6151, DOI 10.17487/RFC6151, March 2011, <<https://www.rfc-editor.org/rfc/rfc6151>>.
- [RFC0760] Postel, J., "DoD standard Internet Protocol", RFC 760, DOI 10.17487/RFC0760, January 1980, <<https://www.rfc-editor.org/rfc/rfc760>>.
- [RFC1958] Carpenter, B., Ed., "Architectural Principles of the Internet", RFC 1958, DOI 10.17487/RFC1958, June 1996, <<https://www.rfc-editor.org/rfc/rfc1958>>.
- [RFC4627] Crockford, D., "The application/json Media Type for JavaScript Object Notation (JSON)", RFC 4627, DOI

10.17487/RFC4627, July 2006, <<https://www.rfc-editor.org/rfc/rfc4627>>.

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

[RFC5704] Bryant, S., Ed., Morrow, M., Ed., and IAB, "Uncoordinated Protocol Development Considered Harmful", RFC 5704, DOI 10.17487/RFC5704, November 2009, <<https://www.rfc-editor.org/rfc/rfc5704>>.

[RFC7159] Bray, T., Ed., "The JavaScript Object Notation (JSON) Data Interchange Format", RFC 7159, DOI 10.17487/RFC7159, March 2014, <<https://www.rfc-editor.org/rfc/rfc7159>>.

[RFC7493] Bray, T., Ed., "The I-JSON Message Format", RFC 7493, DOI 10.17487/RFC7493, March 2015, <<https://www.rfc-editor.org/rfc/rfc7493>>.

[RFC9170] Thomson, M. and T. Pauly, "Long-Term Viability of Protocol Extension Mechanisms", RFC 9170, DOI 10.17487/RFC9170, December 2021, <<https://www.rfc-editor.org/rfc/rfc9170>>.

[SSL3] Barnes, R., Thomson, M., Pironti, A., and A. Langley, "Deprecating Secure Sockets Layer Version 3.0", RFC 7568, DOI 10.17487/RFC7568, June 2015, <<https://www.rfc-editor.org/rfc/rfc7568>>.

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

Acknowledgments

Constructive feedback on this document has been provided by a surprising number of people including Bernard Aboba, Brian Carpenter, Stuart Cheshire, Mark Nottingham, Russ Housley, Eric Rescorla, Henning Schulzrinne, Robert Sparks, Brian Trammell, and Anne Van Kesteren. Please excuse any omission.

Authors' Addresses

Martin Thomson
Mozilla

Email: mt@lowentropy.net

David Schinazi

Google LLC
1600 Amphitheatre Parkway
Mountain View, CA 94043
United States of America

Email: dschinazi.ietf@gmail.com