

**The Harmful Consequences of the Robustness Principle**  
**draft-iab-protocol-maintenance-01**

Abstract

Jon Postel's famous statement of "Be liberal in what you accept, and conservative in what you send" is a principle that 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. For a protocol that is actively maintained, the Postel's robustness principle can, and should, be avoided.

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 April 25, 2019.

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

<a href="#">1.</a>	Introduction . . . . .	<a href="#">2</a>
<a href="#">2.</a>	Fallibility of Specifications . . . . .	<a href="#">3</a>
<a href="#">3.</a>	Protocol Decay . . . . .	<a href="#">4</a>
<a href="#">4.</a>	Ecosystem Effects . . . . .	<a href="#">5</a>
<a href="#">5.</a>	Active Protocol Maintenance . . . . .	<a href="#">6</a>
<a href="#">6.</a>	Extensibility . . . . .	<a href="#">7</a>
<a href="#">7.</a>	The Role of Feedback . . . . .	<a href="#">8</a>
<a href="#">7.1.</a>	Feedback from Implementations . . . . .	<a href="#">8</a>
<a href="#">7.2.</a>	Virtuous Intolerance . . . . .	<a href="#">8</a>
<a href="#">8.</a>	Security Considerations . . . . .	<a href="#">9</a>
<a href="#">9.</a>	IANA Considerations . . . . .	<a href="#">9</a>
<a href="#">10.</a>	Informative References . . . . .	<a href="#">9</a>
<a href="#">Appendix A.</a>	Acknowledgments . . . . .	<a href="#">11</a>
	Author's Address . . . . .	<a href="#">11</a>

## [1.](#) Introduction

Of the great many contributions Jon Postel made to the Internet, his remarkable technical achievements are often shadowed by his contribution of a design and implementation philosophy known as the robustness principle:

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 being the version of the text that appears in IAB [RFC 1958](#) [[PRINCIPLES](#)].

Postel's robustness principle has been hugely influential in shaping the Internet and the systems that use Internet protocols. 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.

Over time, considerable experience has been accumulated with protocols that were designed by the application of Postel's maxim. That experience shows that there are negative long-term consequences to interoperability if an implementation applies Postel's advice.



The flaw in Postel's logic originates from the presumption of an inability to affect change in a system the size of the Internet. That is, once a protocol specification is published, changes that might be different to the practice of existing implementations are not feasible.

Many of the shortcomings that lead to applications of the robustness principle are avoided for protocols under active maintenance. Active protocol maintenance is where a community of protocol designers, implementers, and deployers continuously improve and evolve protocols. A community that takes an active role in the maintenance of protocols can greatly reduce and even eliminate opportunities to apply Postel's guidance.

There is good evidence to suggest that many important protocols are routinely maintained beyond their inception. This document serves primarily as a record of the hazards inherent in applying the robustness principle and to offer an alternative strategy for handling interoperability problems in deployments.

Ideally, protocol implementations never have to apply the robustness principle. Or, where it is unavoidable, any application can be quickly reverted.

## **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 (in [RFC 760](#) [[IP](#)]) 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.

Here Postel recognizes the possibility that the specification could be imperfect. As a frank admission of fallibility it is a significant statement. However, the same statement is inexplicably absent from the later versions in [[HOSTS](#)] and [[PRINCIPLES](#)].

An imperfect specification is natural, largely because it is more important to proceed to implementation and deployment than it is to perfect a specification. A protocol, like any complex system, benefits greatly from experience with its use. A deployed protocol is immeasurably more useful than a perfect protocol.



As [[SUCCESS](#)] demonstrates, success or failure of a protocol depends far more on factors like usefulness than on technical excellence. Postel's timely publication of protocol specifications, even with the potential for flaws, likely had a significant effect in the eventual success of the Internet.

The problem is therefore not with the premise, but with its conclusion: the robustness principle itself.

### **3. Protocol Decay**

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. Application of a protocol to new and unanticipated uses, and ambiguities or errors in the specification are often confounding factors. Situations where two peers disagree on interpretation should be expected over the lifetime of a protocol.

Even with the best intentions, 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 advised by Postel sets up a feedback cycle:

- o Over time, implementations progressively add new code to constrain how data is transmitted, or to permit variations in what is received.
- o Errors in implementations, or confusion about semantics can thereby be masked.
- o 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 Postel's advice, and a product of a natural reluctance to avoid fatal error conditions. Ensuring interoperability in this environment is often colloquially referred to as aiming to be "bug for bug compatible".



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

The original JSON specification [\[JSON\]](#) demonstrates the effect of specification shortcomings. [RFC 4627](#) omitted critical details on a range of key details including Unicode handling, ordering and duplication of object members, and number encoding. Consequently, a range of interpretations were used by implementations. An updated specification [\[JSON-BIS\]](#) did not correct these errors, concentrating instead on identifying the interoperable subset of JSON. I-JSON [\[I-JSON\]](#) 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 reaction that compounds and entrenches interoperability problems.

#### **4. Ecosystem Effects**

Once deviations become entrenched, it can be extremely difficult - if not impossible - to rectify the situation.

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 increases as the number of implementations and bugs increases. Worse, the set of tweaks necessary for interoperability can be difficult to learn.

Consequently, new implementations can be restricted to niche uses, where the problems arising from interoperability issues can be more closely managed. Restricting new implementations to narrow contexts also 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 important in ensuring the continued viability of a protocol. New protocol implementations are also more likely to be developed for new and diverse use cases and often are 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. For instance, an accumulation of mitigations for interoperability issues makes implementations more difficult to maintain.

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 might conceal the problem. That can make it harder, or even impossible, to fix later.

A similar class of problems is described in [RFC 5704](#) [[UNCOORDINATED](#)], which addresses conflict or competition in the maintenance of protocols. This document concerns itself primarily with the absence of maintenance, though the problems are similar.

## 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, the inclination to be tolerant accumulates over time. Indeed, when faced with divergent interpretations of an immutable specification, the best way for an implementation to remain interoperable is to be tolerant of differences in interpretation and an occasional outright implementation error.

From this perspective, application of Postel's advice to the implementation of a protocol specification that does not change is logical, even necessary. But that suggests that the problem is with the assumption that the situation - existing specifications and implementations - are unable to change.

As already established, this is not sustainable. For a protocol to be viable, 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.

Active maintenance of a protocol is critical in ensuring that specifications correctly reflect the requirements for interoperability with existing implementations. Maintenance enables both new implementations and the continued improvement of the protocol. New use cases are an indicator that the protocol could be successful [[SUCCESS](#)].

Protocol designers are strongly encouraged to continue to maintain and evolve protocols beyond their initial inception and definition.



Involvement of protocol implementers is a critical part of this process, as they provide input on their experience with implementation and deployment of the protocol.

Maintenance does not necessarily involve the development of new versions of protocols or protocol specifications. For instance, the most effective means of dealing with a defective implementation in a peer is often to email the developer of the stack. It is far more efficient in the long term to fix one isolated bug than it is to deal with the consequences of workarounds.

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 currency took significant effort.

## **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.

Extensibility 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. If an extension mechanism is designed and implemented correctly, the user of a new protocol feature can confidently predict the effect that feature will have on existing implementations.

Relying on implementations consistently applying 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 variation is preferable in the absence of strong reasons to be flexible.

## **7. The Role of Feedback**

Protocol maintenance is only possible if there is sufficient information about the deployment of the protocol. Feedback from deployment is critical to effective protocol maintenance.

For a protocol specification, the primary and most effective form of feedback comes from people who implement and deploy the protocol. This comes in the form of new requirements, or in experience with the protocol as it is deployed.

Managing and deploying changes to implementations can be expensive. However, it is widely recognized that regular updates are a vital part of the deployment of computer systems for security reasons (see for example [[IOTSU](#)]).

### **7.1. Feedback from Implementations**

Automated error reporting mechanisms in protocol implementations allows for better feedback from deployments. Exposing faults through operations and management systems is highly valuable, but it might be necessary to ensure that the information is propagated further.

Building telemetry and error logging systems that report faults to the developers of the implementation is superior in many respects. However, this is only possible in deployments that are conducive to the collection of this type of information. Giving due consideration to protection of the privacy of protocol participants is critical prior to deploying any such system.

### **7.2. Virtuous Intolerance**

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

Intolerance of any deviation from specification, where implementations generate fatal errors in response to observing undefined or unusal behaviour, can be harnessed to reduce occurrences of abherrent implementations. Choosing to generate fatal error for unspecified conditions instead of attempting error recovery can ensure that faults receive attention.

This improves feedback for new implementations in particular. When a new implementation encounters a virtuously intolerant implementation,



it receives strong feedback that allows problems to be discovered quickly.

To be effective, virtuously intolerant implementations need to be sufficiently widely deployed that they are encountered by new implementations with high probability. This could depend on multiple implementations of the same strict checks. Any intolerance also needs to be strongly supported by specifications, otherwise they encourage fracturing of the protocol community or proliferation of workarounds.

Virtuous intolerance can be used to motivate compliance with any protocol requirement. For instance, the INADEQUATE\_SECURITY error code and associated requirements in HTTP/2 [HTTP2] resulted in improvements in the security of the deployed base.

## **8. 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.

## **9. IANA Considerations**

This document has no IANA actions.

## **10. Informative References**

- [ECMA262] "ECMAScript(R) 2017 Language Specification", ECMA-262 8th Edition, June 2017, <<http://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/info/rfc6709>>.
- [HOSTS] Braden, R., Ed., "Requirements for Internet Hosts - Communication Layers", STD 3, [RFC 1122](#), DOI 10.17487/RFC1122, October 1989, <<https://www.rfc-editor.org/info/rfc1122>>.





- [HTML] "HTML", WHATWG Living Standard, October 2017, <<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/info/rfc7230>>.
- [HTTP2] 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>>.
- [I-JSON] Bray, T., Ed., "The I-JSON Message Format", [RFC 7493](#), DOI 10.17487/RFC7493, March 2015, <<https://www.rfc-editor.org/info/rfc7493>>.
- [IOTSU] Tschofenig, H. and S. Farrell, "Report from the Internet of Things Software Update (IoTSU) Workshop 2016", [RFC 8240](#), DOI 10.17487/RFC8240, September 2017, <<https://www.rfc-editor.org/info/rfc8240>>.
- [IP] Postel, J., "DoD standard Internet Protocol", [RFC 760](#), DOI 10.17487/RFC0760, January 1980, <<https://www.rfc-editor.org/info/rfc760>>.
- [JSON] 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/info/rfc4627>>.
- [JSON-BIS] Bray, T., Ed., "The JavaScript Object Notation (JSON) Data Interchange Format", [RFC 7159](#), DOI 10.17487/RFC7159, March 2014, <<https://www.rfc-editor.org/info/rfc7159>>.
- [PRINCIPLES] Carpenter, B., Ed., "Architectural Principles of the Internet", [RFC 1958](#), DOI 10.17487/RFC1958, June 1996, <<https://www.rfc-editor.org/info/rfc1958>>.
- [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>>.



[TLS] 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>>.

[UNCOORDINATED]

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/info/rfc5704>>.

## **[Appendix A](#). Acknowledgments**

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

Author's Address

Martin Thomson  
Mozilla

Email: [martin.thomson@gmail.com](mailto:martin.thomson@gmail.com)

