

Network Working Group  
Internet-Draft  
Intended status: Informational  
Expires: April 30, 2018

M. Thomson  
Mozilla  
October 27, 2017

## **The Harmful Consequences of the Robustness Principle draft-thomson-postel-was-wrong-02**

### Abstract

Jon Postel's famous statement in [RFC 1122](#) 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, but can produce negative effects in the protocol ecosystem in the long term. Those effects can be avoided by properly maintaining protocols.

### 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 30, 2018.

### Copyright Notice

Copyright (c) 2017 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. Fallibility of Specifications](#) . . . . . [3](#)
- [3. Protocol Decay](#) . . . . . [4](#)
- [4. Ecosystem Effects](#) . . . . . [5](#)
- [5. An Alternative Conclusion](#) . . . . . [6](#)
- [6. The Role of Feedback](#) . . . . . [7](#)
  - [6.1. Fault Reporting is Valuable](#) . . . . . [7](#)
  - [6.2. The Role of Strict Error Handling](#) . . . . . [7](#)
- [7. Implementations Are Ultimately Responsible](#) . . . . . [8](#)
- [8. Security Considerations](#) . . . . . [9](#)
- [9. IANA Considerations](#) . . . . . [9](#)
- [10. Informative References](#) . . . . . [9](#)
- [Appendix A. Acknowledgments](#) . . . . . [10](#)
- [Author's Address](#) . . . . . [10](#)

**1. Introduction**

Of the great many contributions Jon Postel made to the Internet, his remarkable technical achievements are often ignored in favor of the design and implementation philosophy of what is 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](#)].

In comparison, his contributions to the underpinnings of the Internet, which are in many respects more significant, enjoy less conscious recognition. Postel's robustness principle has been hugely influential in shaping the Internet and the systems that use Internet protocols. Many consider this principle to be instrumental in the success of the Internet as well as the design of interoperable protocols in general.

Over time, considerable changes have occurred in both the scale of the Internet and the level of skill and experience available to protocol and software designers. Much of that experience is with



protocols that were designed, informed by Postel's maxim, in the early phases of the Internet.

That experience shows that there are negative long-term consequences to interoperability if an implementation applies Postel's advice. Correcting the problems caused by divergent behavior in implementations can be difficult.

This document describes the negative consequences of the application of Postel's principle to the modern Internet. It recommends against following Postel's advice, instead recommending that protocols receive continuing maintenance after their initial design and deployment. Active maintenance of protocols reduces or eliminates the opportunities to apply Postel's guidance.

There is good evidence to suggest that protocols are routinely maintained beyond their inception. This document serves primarily as a record of the shortcomings of the robustness principle.

## 2. Fallibility of Specifications

What is often missed in discussions of the robustness principle is the context in which it appears. The earliest form of the principle in the RFC series (in [RFC 760](#) [[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 motivating statement is a frank admission of fallibility and remarkable for it. Here Postel recognizes the possibility that the specification could be imperfect. This is an important statement, but inexplicably absent from the later versions in [[HOSTS](#)] and [[PRINCIPLES](#)].

Indeed, an imperfect specification is natural, largely because it was - and remains thus - more important to proceed to implementation and deployment than it is to perfect the protocol specification. A protocol, like software, benefits greatly from experience in deployment, and a deployed protocol is immeasurably more useful than a perfect protocol.

As shown [[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.

In this way 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, in TLS [[RFC5246](#)] messages use an extension format with a tag-length-value format. TLS extensions can be added to handshake messages in any order. However, some server implementations terminate connections if they encounter a TLS ClientHello message



that ends with an empty extension. Thus, client implementations are required to be aware of this incompatibility and ensure that a ClientHello message ends in a non-empty extension.

While TLS highlights the potential for implementation error to cause problems, the original JSON specification [[RFC4627](#)] 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 JSON specification [[RFC7159](#)] was unable to correct these errors, instead concentrating on defining the interoperable subset of JSON. I-JSON [[RFC7493](#)] defines a new format that is substantially similar to JSON while prohibiting the problematic variations. However, that prohibition means that I-JSON is not fully interoperable: an I-JSON implementation will fail to accept some valid JSON texts. Consequently, I-JSON is not widely implemented in parsers. Many JSON parsers instead implement the more precise algorithm specified in [[ECMA262](#)].

It is debatable as to whether decay can be completely avoided, but the robustness principle encourages a reaction that compounds 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. As the set of changes needed to maintain interoperability grows in size, the cost of building a new implementation increases. This is particularly relevant as 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 cannot interoperate, the chances of the addition of further incompatible changes is significant.

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 reduce the ability of established implementations to change. For instance, an accumulation of mitigations for interoperability issues makes implementations more difficult to maintain.

## 5. An Alternative Conclusion

The robustness principle is best suited to safeguarding against flaws in a specification that is intended to remain unchanged for an extended period of time. Indeed, in the face of divergent interpretations of an immutable specification, the only hope for an implementation to remain interoperable is to be tolerant of differences in interpretation and occasional outright implementation errors.

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 presumption of immutability of specifications.

Active maintenance of a protocol can ensure that specifications remain accurate and that new implementations are possible. A maintained protocol is not a static construct, it improves and changes as it is used.

Protocol designers are strongly encouraged to continue to maintain and evolve protocols beyond their initial inception and definition. Maintenance is needed in response to the discovery of errors in specification that might cause interoperability issues. Maintenance is also critical for ensuring that the protocol is viable for application to use cases that might not have been envisaged during its original design. New use cases are an indicator that the protocol could be successful [[SUCCESS](#)].

Maintenance does not demand the development of new versions of protocols or protocol specifications. For instance, [RFC 793](#) [[TCP](#)] remains the canonical TCP reference, but that document alone is no longer sufficient to implement the protocol. TCP is the subject of a very large number of update and extension RFCs that together document the deployed protocol.

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 process of maintenance ideally begins even before the specification for a protocol is complete. 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 the specification to currency took significant effort over more than 6 years.

## **6. 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 or deploy the protocol. An active community of protocol implementers and users is the most valuable source of feedback. It is this community that implements and deploys changes, in addition to contributing to the ongoing maintenance of protocol specifications.

### **6.1. Fault Reporting is Valuable**

Protocol implementations should include automated error reporting mechanisms.

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 consideration to protection of the privacy of protocol participants is critical prior to deploying any such system.

### **6.2. The Role of Strict Error Handling**

Favoring strict error handling over attempting error recovery is an effective technique for ensuring that faults receive attention.

Fatal errors provide excellent motivation to address problems. However, generating fatal errors is only feasible if such errors are sufficiently rare. Frequent problems can result in users ignoring them or finding workarounds, whereas rare events demand greater attention.



Fatal errors or crashes are also preferable if there is any risk that the error might represent an implementation or security issue.

Exposing errors is particularly important for early implementations of a protocol. Enabling stricter error handling helps validate the design of both protocol and implementations. If strict checks are retained when implementations are more widely deployed, those checks better enable the detection and correction of errors in new implementations.

This doesn't mean that protocol implementations need to treat all inputs equally strictly. A protocol could be designed with the explicit goal of accepting a wide range of inputs (see for example [[HTML](#)]). As long as proper reactions to inputs are clearly and unambiguously specified, these considerations don't apply.

## **7. Implementations Are Ultimately Responsible**

Implementations and deployments are critical to the ongoing maintenance of a protocol. Implementations have ample motivation to prefer stability and interoperability over maintenance or correctness. It is likely that - over time - an implementation will accumulate allowances for errors of specification or implementation. Without care, this can lead to suppression of feedback about problems.

Even when an implementation chooses to attempt to adapt to an abnormal condition, communicating that decision to the community is valuable. At a minimum, it allows others to benefit from knowledge of the problem. Discussion about problems might also lead to alternative strategies or protocol enhancements that can avoid further problems.

Over time, mitigations for interoperability issues could become redundant. Occasional review of any special interoperability measures might reveal opportunities for an implementation to switch to stricter handling of exceptional conditions. Removing special allowances in favor of stricter error reporting restores feedback measures that new implementations can then benefit from.

Managing and deploying changes can be expensive. However, it is widely recognized that maintenance is a critical part of the deployment of computer systems for security reasons [[IOTSU](#)]. Managing updates for interoperability problems represents a small additional cost in exchange for ensuring the ability to interoperate with other users of the network.



## **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.

Designers and implementers of security protocols generally understand these concerns. However, general-purpose protocols are not exempt from careful consideration of security issues. Furthermore, because general-purpose protocols tend to deal with flaws or obsolescence in a less urgent fashion than security protocols, there can be fewer opportunities to correct problems in protocols that develop interoperability problems.

## **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>>.
- [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>>.



## [PRINCIPLES]

Carpenter, B., Ed., "Architectural Principles of the Internet", [RFC 1958](#), DOI 10.17487/RFC1958, June 1996, <<https://www.rfc-editor.org/info/rfc1958>>.

[RFC0760] Postel, J., "DoD standard Internet Protocol", [RFC 760](#), DOI 10.17487/RFC0760, January 1980, <<https://www.rfc-editor.org/info/rfc760>>.

[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/info/rfc4627>>.

[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>>.

[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/info/rfc7159>>.

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

[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>>.

## [Appendix A](#). Acknowledgments

Constructive feedback on this document has been provided by a surprising number of people including Mark Nottingham, Brian Trammell, and Anne Van Kesteren. Please excuse any omission.

### Author's Address

Martin Thomson  
Mozilla

Email: martin.thomson@gmail.com

