Network Working Group
Internet-Draft
Intended status: Informational
Expires: September 6, 2018

M. Thomson
Mozilla

March 05, 2018

The Harmful Consequences of the Robustness Principle
draft-thomson-postel-was-wrong-03

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, but can produce negative effects in the protocol ecosystem in the long term. Those effects can be avoided by maintaining protocols.

Table of Contents

1. Introduction 2
2. Fallibility of Specifications 3
3. Protocol Decay 3
4. Ecosystem Effects 5
5. Active Protocol Maintenance 5
6. The Role of Feedback 6
6.1. Error Handling 7
6.2. Feedback from Implementations 7
7. Security Considerations 7
8. IANA Considerations 8
9. Informative References 8
Appendix A. Acknowledgments 9
Author's Address 9

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.

This document shows that flaw in Postel's logic originates from the presumption of immutability of protocol specifications. Thus rather

than apply the robustness principle, this document recommends continuing maintenance for protocols beyond 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 [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.

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 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 in deployment. 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 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:

- Over time, implementations progressively add new code to constrain how data is transmitted, or to permit variations in what is received.

- Errors in implementations, or confusion about semantics can thereby be masked.

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

In this way an 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.

5. Active Protocol Maintenance

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. 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 necessarily involve the development of new versions of protocols or protocol specifications. For instance, RFC 793 [TCP] remains the canonical TCP reference, but a large number of update and extension RFCs together document the protocol as deployed.

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

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 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 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 maintenance is a critical part of the deployment of computer systems for security reasons [IOTSU].

6.1. Error Handling

Ideally, specifications include rules for consistent handling of aberrant conditions as well as expected. This increases the changes that implementations have interoperable handling of unusual conditions.

Choosing to generate fatal error for unspecified conditions instead of attempting error recovery can ensure that faults receive attention. Fatal errors can provide excellent motivation to address a problem if they are sufficiently rare.

A protocol could be designed to permit a narrow set of valid inputs, or it could allow a wide range of inputs (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.

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

7. 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.

8. IANA Considerations

This document has no IANA actions.

9. 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>.

[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>.

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

[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>.

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