Stream: Internet Architecture Board (IAB)
RFC: 9413
Category: Informational
Published: June 2023
ISSN: 2070-1721
Authors: M. Thomson D. Schinazi

RFC 9413
Maintaining Robust Protocols

Abstract

The main goal of the networking standards process is to enable the long-term interoperability of protocols. This document describes active protocol maintenance, a means to accomplish that goal. By evolving specifications and implementations, it is possible to reduce ambiguity over time and create a healthy ecosystem.

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. However, it has been interpreted in a variety of ways. While some interpretations help ensure the health of the Internet, others can negatively affect interoperability over time. When a protocol is actively maintained, protocol designers and implementers can avoid these pitfalls.

Status of This Memo

This document is not an Internet Standards Track specification; it is published for informational purposes.

This document is a product of the Internet Architecture Board (IAB) and represents information that the IAB has deemed valuable to provide for permanent record. It represents the consensus of the Internet Architecture Board (IAB). Documents approved for publication by the IAB are not candidates for any level of Internet Standard; see Section 2 of RFC 7841.

Information about the current status of this document, any errata, and how to provide feedback on it may be obtained at https://www.rfc-editor.org/info/rfc9413.

Copyright Notice

Table of Contents

1. Introduction

There is good evidence to suggest that many important protocols are routinely maintained beyond their inception. In particular, a sizable proportion of IETF activity is dedicated to the stewardship of existing protocols. This document first discusses hazards in applying the robustness principle too broadly (see Section 2) and offers an alternative strategy for handling interoperability problems in deployments (see Section 5).

Ideally, protocol implementations can be actively maintained so that unexpected conditions are proactively identified and resolved. Some deployments might still need to apply short-term mitigations for deployments that cannot be easily updated, but such cases need not be permanent. This is discussed further in Section 5.

2. Protocol Robustness

The robustness principle has been hugely influential in shaping the design of the Internet. As stated in the IAB document "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.

There are three main aspects to the robustness principle:

Robustness to software defects: No software is perfect, and failures can lead to unexpected behavior. Well-designed software strives to be resilient to such issues, whether they occur in the local software or in software that it communicates with. In particular, it is critical for software to gracefully recover from these issues without aborting unrelated processing.

Robustness to attacks: Since not all actors on the Internet are benevolent, networking software needs to be resilient to input that is intentionally crafted to cause unexpected consequences. For example, software must ensure that invalid input doesn't allow the sender to access data, change data, or perform actions that it would otherwise not be allowed to.

Robustness to the unexpected: It can be possible for an implementation to receive inputs that the specification did not prepare it for. This scenario excludes those cases where a the specification explicitly defines how a faulty message is handled. Instead, this refers to cases where handling is not defined or where there is some ambiguity in the specification. In this case, some interpretations of the robustness principle advocate that the implementation tolerate the faulty input and silently discard it. Some interpretations even suggest that a faulty or ambiguous message be processed according to the inferred intent of the sender.

The facets of the robustness principle that protect against defects or attacks are understood to be necessary guiding principles for the design and implementation of networked systems. However, an interpretation that advocates for tolerating unexpected inputs is no longer considered best practice in all scenarios.

Time and experience show that negative consequences to interoperability accumulate over time if implementations silently accept faulty input. This problem originates from an implicit assumption that it is not possible to effect change in a system the size of the Internet. When one assumes that changes to existing implementations are not presently feasible, tolerating flaws feels inevitable.

Many problems that this third aspect of 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.

2.1. 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 a 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 "What Makes for a Successful Protocol?" [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, ignoring faulty or ambiguous input is almost always the incorrect solution to the problem.

2.2. 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 new types of input.

A well-designed extensibility mechanism establishes clear rules for the handling of elements like new messages or parameters. This depends on specifying the handling of malformed or illegal inputs so that implementations behave consistently in all cases that might affect interoperation. New messages or parameters thereby become entirely expected. 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 handle unexpected input 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]. It is better and easier to design a protocol for extensibility initially than to retrofit the capability (see also [EDNS0]).

2.3. Flexible Protocols

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

A more flexible protocol is more complex to specify and implement; variations, especially those that are not commonly used, can create potential interoperability hazards. In the absence of strong reasons to be flexible, a simpler protocol is more likely to successfully interoperate.

Where input is provided by users, allowing flexibility might serve to make the protocol more accessible, especially for non-expert users. HTML authoring [HTML] is an example of this sort of design.

In protocols where there are many participants that might generate messages based on data from other participants, some flexibility might contribute to resilience of the system. A routing protocol is a good example of where this might be necessary.

In BGP [BGP], a peer generates UPDATE messages based on messages it receives from other peers. Peers can copy attributes without validation, potentially propagating invalid values. RFC 4271 [BGP] mandated a session reset for invalid UPDATE messages, a requirement that was not widely implemented. In many deployments, peers would treat a malformed UPDATE in less stringent ways, such as by treating the affected route as having been withdrawn. Ultimately, RFC 7606 [BGP-REH] documented this practice and provided precise rules, including mandatory actions for different error conditions.

A protocol can explicitly allow for a range of valid expressions of the same semantics, with precise definitions for error handling. This is distinct from a protocol that relies on the application of the robustness principle. With the former, interoperation depends on specifications that capture all relevant details, whereas interoperation in the latter depends more extensively on implementations making compatible decisions, as noted in Section 4.2.

3. Applicability

The guidance in this document is intended for protocols that are deployed to the Internet. There are some situations in which this guidance might not apply to a protocol due to conditions on its implementation or deployment.

In particular, this guidance depends on an ability to update and deploy implementations. Being able to rapidly update implementations that are deployed to the Internet helps manage security risks, but in reality, some software deployments have lifecycles that make software updates either rare or altogether impossible.

Where implementations are not updated, there is no opportunity to apply the practices that this document recommends. In particular, some practices -- such as those described in Section 5.1 -- only exist to support the development of protocol maintenance and evolution. Employing this guidance is therefore only applicable where there is the possibility of improving deployments through timely updates of their implementations.

4. Harmful Consequences of Tolerating the Unexpected

Problems in other implementations can create an unavoidable need to temporarily tolerate unexpected inputs. However, this course of action carries risks.

4.1. Protocol Decay

Tolerating unexpected input might be an expedient tool for systems in early phases of deployment, which was the case for the early Internet. Being lenient in this way 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 tolerating the unexpected 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 with these servers, which were widely deployed, client implementations were required to be aware of this bug and ensure that a ClientHello message ends in a non-empty extension.

Overapplication of the robustness principle therefore encourages a chain reaction that can create interoperability problems over time. In particular, tolerating unexpected behavior is particularly deleterious for early implementations of new protocols, as quirks in early implementations can affect all subsequent deployments.

4.2. 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 over time.
- Conversely, if non-compliance is tolerated by existing implementations, non-compliant implementations can be deployed successfully. Newer implementations then have a 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 test suites -- where they exist -- can guide the initial development of implementations. Ultimately, the need to interoperate with deployed implementations is a de facto conformance test suite that can supersede any formal protocol definition.

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 "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 specifications and then deploying those changes, makes the protocol better. Tolerating unexpected input instead conceals problems, making it harder, if not 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, the 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. However, when official specifications fail to be updated, then deployed implementations -- including their quirks -- often become a substitute standard.

Tolerating unexpected inputs from another implementation might seem logical, even necessary. However, 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. For example, when an implementer discovers a scenario where a specification defines some input as faulty but does not define how to handle that input, the implementer can provide significant value to the ecosystem by reporting the issue and helping to evolve the specification.

When a discrepancy is found between a specification and its implementation, a maintenance discussion inside the standards process allows reaching consensus on how best to evolve the specification. Subsequently, updating implementations to match evolved specifications 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 evolves in concert with implementations. 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, as software defects can happen even when the specification is unambiguous. 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.

5.1. 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 so that they are encountered by new implementations with high probability. This could depend on multiple implementations deploying strict checks.

Interoperability problems also need to be made known to those in a position to address them. In particular, systems with human operators, such as user-facing clients, are ideally suited to surfacing errors. Other systems might need to use less direct means of making errors known.

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 2.2.

Any intolerance also needs to be strongly supported by specifications; otherwise, they encourage fracturing of the protocol community or proliferation of workarounds. See Section 5.2.

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.

A notification for a fatal error is best sent as explicit error messages to the entity that made the error. Error messages benefit from being able to carry arbitrary information that might help the implementer of the sender of the faulty input understand and fix the issue in their software. QUIC error frames [QUIC] are an example of a fatal error mechanism that helped implementers improve software quality throughout the protocol lifecycle. Similarly, the use of Extended DNS Errors [EDE] has been effective in providing better descriptions of DNS resolution errors to clients.

Stateless protocol endpoints might generate denial-of-service attacks if they send an error message in response to every message that is received from an unauthenticated sender. These implementations might need to silently discard these messages.

5.2. 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 backward-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.

Accidentally excluding unexpected participants is not usually a good outcome. When developing and deploying changes, it is best to first understand the extent to which the change affects existing deployments. This ensures that any exclusion that occurs is intentional.

In some cases, existing deployments might need to change in order to avoid being excluded. Though it might be preferable to avoid forcing deployments to change, this might be considered necessary. To avoid unnecessarily excluding deployments that might take time to change, developing a migration plan can be prudent.

Exclusion is a direct goal when choosing to be intolerant of errors (see Section 5.1). 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 that might arise from tolerating the unexpected. The IAB document "Uncoordinated Protocol Development Considered Harmful" [RFC5704] describes how conflict or competition in the maintenance of protocols can lead to similar problems.

6. Security Considerations

Careless 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, weak primitives [MD5] and obsolete mechanisms [SSL3] are good examples of the use of unsafe security practices where forcing exclusion (Section 5.2) can be desirable.

7. IANA Considerations

This document has no IANA actions.

8. Informative References

[BGP] Rekhter, Y., Ed., Li, T., Ed., and S. Hares, Ed., "A Border Gateway Protocol 4 (BGP-4)", RFC 4271, DOI 10.17487/RFC4271, January 2006, <https://www.rfc-editor.org/info/rfc4271>.

[BGP-REH] Chen, E., Ed., Scudder, J., Ed., Mohapatra, P., and K. Patel, "Revised Error Handling for BGP UPDATE Messages", RFC 7606, DOI 10.17487/RFC7606, August 2015, <https://www.rfc-editor.org/info/rfc7606>.

[EDE] Kumari, W., Hunt, E., Arends, R., Hardaker, W., and D. Lawrence, "Extended DNS Errors", RFC 8914, DOI 10.17487/RFC8914, October 2020, <https://www.rfc-editor.org/info/rfc8914>.

[EDNS0] Damas, J., Graff, M., and P. Vixie, "Extension Mechanisms for DNS (EDNS(0))", STD 75, RFC 6891, DOI 10.17487/RFC6891, April 2013, <https://www.rfc-editor.org/info/rfc6891>.

[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>.

[HTML] WHATWG, "HTML - Living Standard", <https://html.spec.whatwg.org/>.

[HTTP] Fielding, R., Ed., Nottingham, M., Ed., and J. Reschke, Ed., "HTTP Semantics", STD 97, RFC 9110, DOI 10.17487/RFC9110, June 2022, <https://www.rfc-editor.org/info/rfc9110>.

[HTTP/2] Thomson, M., Ed. and C. Benfield, Ed., "HTTP/2", RFC 9113, DOI 10.17487/RFC9113, June 2022, <https://www.rfc-editor.org/info/rfc9113>.

[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/info/rfc6151>.

[QUIC] Iyengar, J., Ed. and M. Thomson, Ed., "QUIC: A UDP-Based Multiplexed and Secure Transport", RFC 9000, DOI 10.17487/RFC9000, May 2021, <https://www.rfc-editor.org/info/rfc9000>.

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

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

[RFC3117] Rose, M., "On the Design of Application Protocols", RFC 3117, DOI 10.17487/RFC3117, November 2001, <https://www.rfc-editor.org/info/rfc3117>.

[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/info/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/info/rfc5704>.

[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/info/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/info/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/info/rfc8446>.

IAB Members at the Time of Approval

Internet Architecture Board members at the time this document was approved for publication were:

Jari Arkko

Deborah Brungard

Lars Eggert

Wes Hardaker

Cullen Jennings

Mallory Knodel

Mirja Kühlewind

Zhenbin Li

Tommy Pauly

David Schinazi

Russ White

Qin Wu

Jiankang Yao

The document had broad but not unanimous approval within the IAB, reflecting that while the guidance is valid, concerns were expressed in the IETF community about how broadly it applies in all situations.

Acknowledgments

Constructive feedback on this document has been provided by a surprising number of people including, but not limited to, the following: Bernard Aboba, Brian Carpenter, Stuart Cheshire, Joel Halpern, Wes Hardaker, Russ Housley, Cullen Jennings, Mallory Knodel, Mirja Kühlewind, Mark Nottingham, Eric Rescorla, Henning Schulzrinne, Job Snijders, Robert Sparks, Dave Thaler, Brian Trammell, and Anne van Kesteren. Some of the properties of protocols described in Section 4.1 were observed by Marshall Rose in Section 4.5 of [RFC3117].

Authors' Addresses

Martin Thomson
Email: mt@lowentropy.net

David Schinazi
Email: dschinazi.ietf@gmail.com