Network Working Group
Internet-Draft
Intended status: Informational
Expires: 3 September 2026

L. Pardue
Cloudflare
T. Pauly
Apple
D. Thaler
Armidale Consulting

2 March 2026

Considerations For Maintaining Protocols Using Grease and Variability
draft-iab-protocol-greasing-00

Abstract

Active use and maintenance of network protocols is an important way to ensure that protocols remain interoperable and extensible over time. Techniques such as intentionally exercising extension points with non-meaningful values (referred to as "grease") or adding variability to how protocol elements are used help generate this active use.

Grease and variability are used across various protocols developed by the IETF. This document discusses considerations when designing and deploying grease and variability mechanisms, and provides advice for making them as effective as possible.

About This Document

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

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

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

Status of This Memo

Copyright Notice

Table of Contents

1. Introduction

Section 3 of [VIABILITY] discusses "active use" as a category of techniques that protocol designers and implementers employ to ensure that protocol extension mechanisms are exercised and can be used in the future. This ability to change (to handle protocol updates and extensions) is an important factor in the success of protocol deployment, as discussed in [SUCCESS].

Active use of protocol features and extensions often requires intentional efforts beyond what would organically occur in deployments. Some extension points do not frequently see new values being used, but that they remain usable in the future is important. Some patterns of protocol usage might be relatively static without specific efforts to ensure that they can change in the future.

One key technique for intentional use is "grease" (an acronym for Generate Random Extensions And Sustain Extensibility), or "greasing". Greasing was initially designed for TLS [GREASE] and was later adopted by other protocols such as QUIC [QUIC]. In these protocols, extension codepoints are reserved only for greasing and must be ignored upon receipt. Greasing is suitable for many protocols but not all; Section 3.3 of [VIABILITY] discusses the applicability and limitations of greasing.

While it is becoming more common, designing and applying grease is not necessarily trivial. There are best practices, and some common pitfalls to avoid, that have been developed by the protocols using grease thus far. Section 2 takes these learnings and provides considerations for new protocol design and deployment.

Separate from greasing using reserved values, protocol deployments can intentionally add variability to ensure that network endpoints and middleboxes do not end up ossifying certain patterns. For example, an HTTP deployment focused on downloads might want to add support for uploads. Changing use of the application and transport protocol features can affect the deployment’s network traffic profile. If expectations have been formed around historical patterns of use, i.e., ossification, introducing change might lead to deployment problems. Section 4 presents considerations about how intentionally increasing the variability of protocols can mitigate some of these concerns.

Protocol extensions can provide longevity in the face of changing needs or environment. However, a replacment protocol might be preferred when extensions are not adequate or feasible. A protocol replacement could aggregate common extensions and possibly make them mandatory, effectively defining a new baseline that can simplify deployment and interoperability. A replacement protocol version may or may not be compatible with other versions. A protocol may or may not have a mechanism for version selection or agility. Section 5 presents considerations about designing for and/or implementing version negotiation and migration.

2. Considerations for Greasing Protocols

Greasing can take many forms, depending on the protocol and the nature of its extension points. The common pattern across forms of greasing is that values are generated that have no useful meaning to the protocol and are meant to be ignored upon receipt. Such values used for the purpose of greasing are referred to as "grease values" within this document.

More background to this approach is given in Section 3.3 of [VIABILITY].

This section provides some practical considerations for how to define and use greasing, and avoid possible pitfalls.

2.1. Define and Register Grease Value Ranges

Many protocols that use greasing have a limited set of possible values or codepoints that can be used in a particular extension point. A common approach is to reserve a subset of the registrable space for greasing.

The following are some examples of protocols that have reserved codepoints for grease values:

* TLS ([GREASE])
* QUIC (Section 18.1 of [QUIC])
* HTTP/3 (Section 7.2.8 of [HTTP/3])
* Privacy Pass ([PRIVACYPASS])

The specifics of how to reserve values depends on the nature of the available space. For protocols with large possible spaces, it is useful to have a large set of grease values to increase the chance that receiver greasing requirements are exercised. The specific size and distribution of the grease range needs to accommodate the protocol constraints. For instance, protocols that use 8-bit fields may find it too costly to dedicate many grease values, while 32-bit or 64-bit fields are likely to have no such limitations.

It is recommended to use an algorithm to reserve large sets of values. For example, [QUIC] uses an algorithm of 31 * N + 27 to allocate grease values for transport parameters.

One possible problem with some algorithms is that they will spread out values over the space, and impact the ability to use or reserve contiguous blocks of non-grease values. It is common for protocol extension designers to want to reserve contiguous blocks of codepoints in order to aid iteration and experimentation. Reserved grease values can end up being in spaces that would otherwise be used for such contiguous blocks.

Codepoints being used for new reservations, or experimentation, need to be careful to not unintentionally use grease values. Doing so could lead to interoperability failures.

2.1.1. Recommendations for IANA Considerations

IANA registries that contain reserved grease values need to indicate that the values are reserved in order to prevent them from being allocated for other uses. The specifics of how to represent the reservations is up to the documents that define the registries.

Some registries list out the reserved grease values individually, marked as "Reserved". For example, the TLS registry uses this approach (https://www.iana.org/assignments/tls-parameters/tls-parameters.xhtml).

If an algorithm or pattern is used to define grease values, it is recommended when possible to instead only define a single entry for the entire grease value set. This entry should include the pattern or algorithm. This approach is used by the QUIC registry (https://www.iana.org/assignments/quic/quic.xhtml) and the HTTP/3 registry (https://www.iana.org/assignments/http3-parameters/http3-parameters.xhtml#http3-parameters-frame-types)

Grease values must not be used or registered for any other purpose. This is in contrast to other potentially "reserved" values that might be reused or claimed for a new purpose in the future. To avoid confusion, it is recommended to label the reservation with a clear identifier, such as "reserved for greasing".

2.2. Use Unpredictable Grease Values

In order to gain the benefits of active use and avoid ossification, grease values used in actual packets need to be sent in ways that won’t become a predictable pattern that receiver and middlebox implementations and deployments ossify around.

Implementations that generate grease values should pick unpredictable entries from the set of reserved grease values. It is most important that values be unpredictable across the set of all protocol

participants for particular deployments. This can be achieved in multiple ways: for example, an individual sender might pick random values from the grease value space on each interaction; alternatively, a single sender could pick a specific grease value to use, while other senders pick other values.

In order to support picking unpredictable values, the set of reserved values should be large, when possible. See Section 2.1 for a discussion of how to allocate grease values.

2.3. Use Grease Values Unpredictably

In addition to selecting unpredictable values, the inclusion of grease itself can be made unpredictable. Implementations can vary their behavior by including no grease values, one grease value, or multiple grease values for a given protocol extension point.

How consistently and frequently to use grease values is a choice that implementations and deployments need to consider and weigh against several factors.

Deployments of greasing should consider how they expect errors exposed by using grease values to be noticed and measured.

If grease values are sent too infrequently, so that errors due to sending grease values blend in with the noise of other errors, it is likely that no one will notice failures, thus defeating the purpose of greasing. When grease values are sent more frequently, they will be noticed more. However, if grease values are sent too consistently, receiver implementations might end up special-casing grease values.

The patterns for sending grease values can be made more effective by coordinating between devices sending the values. One example of coordination is having a "flag day" where implementations start sending grease values broadly, and measure to see where errors occur. See Section 3 of [TRANSITION] for more considerations concerning the use of a flag day.

2.4. Don’t Handle Grease Values as a Special Case

Implementations that read and process grease values must ignore the values. "Ignoring" a value upon receipt can have multiple dimensions, however. Simply not performing any protocol action based on the grease value isn’t enough to ensure that the protocol will remain extensible. The ignoring must be handled as a general case of "unknown" or "unhandled" values, not as a special case for ignored grease values.

This means that grease values can only meaningfully be used for protocol elements where all unknown values are ignored by default. (Protocols may have ways to indicate that some specific values are "mandatory" or "critical" in order to have the protocol interaction succeed, but these must not be marked for any grease values.)

One pitfall that implementations may encounter when building logic to handle the receipt of grease values is related to cases where some recognized non-grease values need to be handled as errors. Consider the following abstract example:

1. A protocol element has values 1, 2, 3, and 4 defined and registered for use. The values 13 and 42 are reserved as grease values.

2. In a specific scenario, only the known values 1 and 2 are valid; 3 and 4 are considered errors.

3. An implementation might naively choose to check for the value being 1 or 2, handle those cases, and send an error otherwise.

4. When grease values are used, the previous logic will flag an error for the grease value. If this is detected, implementations might choose to work around this by updating the logic to check for the value being 1 or 2, then check for grease values to ignore, and then send an error otherwise. This logic is also incorrect since it doesn’t allow for new extensibility.

The correct logic for the above scenario would be to check for the value being 1 or 2, then explicitly check for the value being 3 or 4 (to handle the error), with a catch-all for ignored values.

In pseudo-code, the correct logic would work like this, where the grease values would fall into the final else case as ignored values.

```
if is_valid_case_one(value):
  handle_case_one()
else if is_valid_case_two(value):
  handle_case_two()
else if is_known_invalid_case(value):
  handle_error()
else:
  ignore_value()
```

Implementations need to take care when implementing such logic. Protocol specification designers should emphasize that grease values must not be special-cased. It is also recommended to provide example logic or pseudocode in specifications, similar to the example above,

as guidance to implementers on how to correctly process protocol elements like these. Documents can also provide test vectors, when applicable, that include grease values to ensure they are processed correctly.

One limitation of greasing is that it only exercises grease values. It does not check whether other values that are reserved for future use are correctly treated as ignored or errors. As such, ossification remains a possibility for non-grease values. The goal of greasing is to increase the chances of correct implementation, and thereby reduce (but not eliminate) the possibility of ossification.

3. Deployment Considerations and Incentives for Greasing

Greasing can be used as a tool to improve the active use of existing protocol elements (which weren't necessarily designed with greasing to begin with, or weren't deployed with greasing); or as part of new protocol design and deployments.

When greasing isn't used from the beginning of protocol deployment, starting to use greasing comes with the risk of triggering failures or anomalies. These failures might be innocuous, but they also might be very impactful and visible to users. This risk creates a disincentive to deploy greasing in existing systems, since generally the change that triggers failures is often blamed for the failure. The risk is highest when adding greasing to a particular protocol flow that doesn't require any change of behavior or adoption to hit the greasing behavior. For example, if a service migrates to use a new web server implementation that enables greasing, while the previous server didn't, some new failures may be hit if clients react poorly to greasing.

Some approaches to avoid failures due to greasing include:

* Designing, implementing, and using greasing very early on in protocol development and deployment. This avoids the aforementioned risk of adding greasing late in a deployment.

* Enabling greasing along with other major protocol feature changes or deployment changes. For example, when upgrading to a new protocol version that requires implementation updates on multiple systems, greasing can be added for the new version specifically. This approach works well for situations where the protocol participants are known and already need to cooperate (such as within an encrypted protocol between two endpoints). This approach applies less well for situations where non-cooperating entities (such as middleboxes) are the source of ossification.

* Using heuristics to disable greasing when errors are encountered. For example, if a client-initiated protocol operation fails multiple times when grease values are used, it can be retried without any grease values. Alternatively, if a server recognizes a property of a client that always fails when greasing is used, it could choose to disable greasing when that client is detected. This reduces the effectiveness of grease values in removing existing ossification, but can still have benefits for flagging issues in new implementations when they receive grease values.

3.1. Discouraging Ossification

Greasing alone can help ensure that middleboxes tolerate unrecognized extensions rather than crashing, but one common middlebox behavior is to simply drop all unrecognized extensions. Techniques such as disabling greasing when errors are encountered can enable communication but may not provide enough incentive to discourage ossification.

One approach to preserving the incentive to grease without being blamed for doing so, while still potentially discouraging ossification, is to make any such ossification visible. For example, a sender can log and report ossification issues on a given path when falling back to non-grease values. Similarly, a receiver can log and report communication that does not include grease values. Such reports might be used in social feedback (e.g., public product or service reviews), or in contractual business relationship (e.g., SLA) discussion, in order to provide an incentive to implementers or deployers of ossified implementations.

Similar techniques can be used to encourage greasing and variability by others, to provide further discouragements against ossification in the future. Having a receiver log and report communication that does not include grease values, as noted above, can also be used to report on senders that do not support greasing. It is useful, however, to be able to distinguish between the sender not supporting greasing, versus the sender supporting greasing but only non-greased messages arriving at the receiver. Protocols defining greasing mechanisms should consider whether receivers can distinguish between these two cases.

It is not generally recommended to use more drastic incentives to require greasing such as degrading performance or reliability for senders that don’t support greasing. While this is possible, the benefits of greasing are longer term, rather than immediate, and the incentives are often not aligned. Section 2.1.1 of [SUCCESS] explains that success most easily comes when those who are required to make a change are the ones who gain the most benefit, and the

benefit is noticeable. If the sending implementation does not plan to use new extensions in the future, it would not see any benefit from being pushed to add greasing.

4. Considerations for Increasing Protocol Variability

Greasing can maintain protocol extensibility by falsifying active use of its extension points (see Section 3.3 of [VIABILITY]). However, greasing alone does not ensure positive use of extension mechanisms. A protocol may define a wide-ranging extension capability that remains unused in the absence of real use cases. This can lead to ossification that does not expect extensions, leading to interoperability problems later on.

Long-term maintenance and interoperability can be ensured by exercising extension points positively. To some extent this can be thought of as protocol fuzzing. This might be difficult to exercise because varying the protocol elements might change the outcome of interactions, leading to real errors. However, some protocols allow elements to be safely changed, as shown in the following examples. The principles in these examples are not limited to the protocols mentioned, but also arise in many other protocols as well (e.g., the Session Initiation Protocol (SIP) [SIP]).

4.1. Example: QUIC frames

QUIC packets contain frames. Receivers might build expectations on the longitudinal aspects of packets or frames - size, ordering, frequency, etc. A sender can quite often manipulate these parameters and stay compliant to the requirements of the QUIC protocol.

A QUIC stream is an ordered reliable byte stream that is serialized as a sequence of STREAM frames with a length and offset. Receivers are expected to reassemble frames, which could arrive in any order, into an ordered reliable byte stream that is readable by applications.

A form of positive testing is for a sender to unpredictably order the STREAM frames that it transmits. For example, the sender can vary the sequence order of offset values. This allows exercising the QUIC reassembly features of the receiver with the expectation that no failure would occur. However, doing this may introduce delay or stream head-of-line blocking that affects the performance aspects of a transmission, which may not be acceptable for a given use case. As such, positive testing might be most appropriate to use in a subset of connections, or phases within a connection.

4.2. Example: Post-quantum key exchange in TLS

Post-quantum key exchange in TLS (such as defined in [PQTLS]) expands the key sizes sent in Client Hello messages in TLS 1.3 [TLS]. Before using these algorithms, Client Hello messages would generally fit within a single packet (either in TCP or QUIC). However, with these larger keys, Client Hello messages need to be split across two (or more) separate packets. Initial deployments of these keys uncovered many buggy server and middlebox implementations that did not correctly handle Client Hello messages being split across multiple packets.

This is a case in which adding variability to how TLS Client Hello messages were sent could have helped avoid ossification and buggy implementations. Variations could include intentionally splitting the messages across packets without increasing message size, and also adding other values to messages to force them to exceed the length that can fit within a single packet.

4.3. Example: IPv6

The IPv6 protocol [IPV6] defines the ability to use extension headers, and further defines the ability to include options in two of those headers. It also defines a recommended (but not required) order of such headers.

If the same headers tend to appear in the same order, and with the same size and offset most of the time, middleboxes can begin to incorrectly rely on assumptions that those will always be the case, and lead to ossification and inability to use new headers or options or differing orders in the future.

IPv6 did not define grease values, but a sender may vary some aspects of the protocol, such as varying the order of options within an extension header, or even including a Destination Options Header with only Pad1 or PadN options between other headers.

While doing such variations can discourage ossification, they might also have a negative performance impact if done indiscriminately. For example, making packets larger as a result might result in fragmentation.

Finally, there may be security or gateway middleboxes already deployed that incorrectly rely on assumptions about header order, size, or location in a packet. Introducing variability might then cause operational problems in such networks and result in a disincentive to use the protocol or device or application using it.

5. Considerations for Protocol Versions

[TRANSITION] discusses considerations around planning for transitioning from an existing protocol, or protocol version, to a new one. There are also intrinsic and well-documented issues related to testing version negotiation of protocols; see [EXTENSIBILITY] and Sections 2.1 and 3.2 of [VIABILITY].

One way to grease protocol versions is to have a protocol pass a list of supported versions or features (e.g., cipher suites), along with a grease value, such that the grease value will not impact the actual version or features chosen, since it will not be selected by the receiving entity.

Another method is to have a protocol include a recovery mechanism (e.g., an extra round trip to try with another option) for cases when an unsupported version or feature is attempted. In this case, a grease value might be attempted at some frequency or opportunity that would not adversely affect performance.

6. Security Considerations

The considerations in [MAINTENANCE], [GREASE], [END-USERS], and [VIABILITY] all apply to the topics discussed in this document.

The use of protocol features, extensions, and versions can already allow fingerprinting [PRIVCON]. Any techniques that change parameters in any way, including but not limited to those discussed in this document, can affect fingerprinting. A deeper analysis of this topic has been deemed out of scope.

7. IANA Considerations

This document has no IANA actions itself. Guidance on how other documents can effectively instruct IANA about protocol greasing is provided in Section 2.1.1

8. Informative References

[END-USERS]
Nottingham, M., "The Internet is for End Users", RFC 8890, DOI 10.17487/RFC8890, August 2020, <https://www.rfc-editor.org/rfc/rfc8890>.

[EXTENSIBILITY]
Carpenter, B., Aboba, B., Ed., and S. Cheshire, "Design Considerations for Protocol Extensions", RFC 6709, DOI 10.17487/RFC6709, September 2012, <https://www.rfc-editor.org/rfc/rfc6709>.

[GREASE] Benjamin, D., "Applying Generate Random Extensions And Sustain Extensibility (GREASE) to TLS Extensibility", RFC 8701, DOI 10.17487/RFC8701, January 2020, <https://www.rfc-editor.org/rfc/rfc8701>.

[HTTP/3] Bishop, M., Ed., "HTTP/3", RFC 9114, DOI 10.17487/RFC9114, June 2022, <https://www.rfc-editor.org/rfc/rfc9114>.

[IPV6] Deering, S. and R. Hinden, "Internet Protocol, Version 6 (IPv6) Specification", STD 86, RFC 8200, DOI 10.17487/RFC8200, July 2017, <https://www.rfc-editor.org/rfc/rfc8200>.

[MAINTENANCE]
Thomson, M. and D. Schinazi, "Maintaining Robust Protocols", RFC 9413, DOI 10.17487/RFC9413, June 2023, <https://www.rfc-editor.org/rfc/rfc9413>.

[PQTLS] Kwiatkowski, K., Kampanakis, P., Westerbaan, B., and D. Stebila, "Post-quantum hybrid ECDHE-MLKEM Key Agreement for TLSv1.3", Work in Progress, Internet-Draft, draft-ietf-tls-ecdhe-mlkem-04, 8 February 2026, <https://datatracker.ietf.org/doc/html/draft-ietf-tls-ecdhe-mlkem-04>.

[PRIVACYPASS]
Pauly, T., Valdez, S., and C. A. Wood, "The Privacy Pass HTTP Authentication Scheme", RFC 9577, DOI 10.17487/RFC9577, June 2024, <https://www.rfc-editor.org/rfc/rfc9577>.

[PRIVCON] Cooper, A., Tschofenig, H., Aboba, B., Peterson, J., Morris, J., Hansen, M., and R. Smith, "Privacy Considerations for Internet Protocols", RFC 6973, DOI 10.17487/RFC6973, July 2013, <https://www.rfc-editor.org/rfc/rfc6973>.

[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/rfc/rfc9000>.

[SIP] Rosenberg, J., Schulzrinne, H., Camarillo, G., Johnston, A., Peterson, J., Sparks, R., Handley, M., and E. Schooler, "SIP: Session Initiation Protocol", RFC 3261, DOI 10.17487/RFC3261, June 2002, <https://www.rfc-editor.org/rfc/rfc3261>.

[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/rfc/rfc5218>.

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

[TRANSITION] Thaler, D., Ed., "Planning for Protocol Adoption and Subsequent Transitions", RFC 8170, DOI 10.17487/RFC8170, May 2017, <https://www.rfc-editor.org/rfc/rfc8170>.

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

Acknowledgments

This work is a summary of the topics discussed during EDM meetings. The contributors at those meetings are thanked.

Authors' Addresses

Lucas Pardue
Cloudflare
Email: lucas@lucaspardue.com

Tommy Pauly
Apple
Email: tpauly@apple.com

Dave Thaler
Armidale Consulting
Email: dave.thaler.ietf@gmail.com