Network Working Group
Internet-Draft
Intended status: Informational
Expires: 9 January 2020

S. McQuistin
V. Band
C. Perkins
University of Glasgow
8 July 2019

Fully Specifying Protocol Parsing with Augmented ASCII Diagrams
draft-mcquistin-augmented-ascii-diagrams-00

Abstract

This document describes a machine-readable format for fully specifying the process by which a protocol can be parsed. This format combines a consistent ASCII packet diagram format with the use of structured text, maintaining human readability while enabling support for machine parsing. This document is itself an example of how this format can be used.

Status of This Memo

Copyright Notice

Table of Contents

1. Introduction

ASCII packet header diagrams have become the de-facto format for describing the syntax of binary protocols. In otherwise largely textual documents, they allow for the visualisation of packet formats, reducing human error, and aiding in the implementation of parsers for the protocols that they specify. Given their widespread use, and relatively structured form, ASCII packet header diagrams provide a good base from which to develop a format that supports the automatic generation of parser code from protocol standards documents.

There are two broad issues with the existing ASCII packet header diagrams that need to be addressed to enable machine-readability. First, their use, while sufficiently consistent for human readability, contains enough variation to make machine parsing difficult: different documents tend to use subtly different formats and conventions. Second, ASCII packet header diagrams alone do not fully capture the parsing process for protocols, requiring supplementary text. To support machine parsing, this supplementary text must be consistently structured.

This document describes a consistent ASCII packet header format and accompanying structured text constructs that allow for the parsing

process of protocol headers to be fully specified. This provides support for the automatic generation of parser code. Broad design principles, that seek to maintain the primacy of human readability and flexibility in authorship, are described, before the format itself is given.

This document is itself an example of the approach that it describes, with the ASCII packet diagrams and structured text format described by example.

This draft describes early work. As consensus builds around the particular syntax of the format described, both a formal ABNF specification and code that parses it (and, as described above, this document) will be provided.

2. Background

This section begins by considering how ASCII packet header diagrams are used in existing documents. This exposes the limitations that the current usage has in terms of machine-readability, guiding the design of the format that this document proposes.

While this document focuses on the machine-readability of packet header diagrams, this section also discusses the use of other structured or formal languages within IETF documents. Considering how and why these languages are used provides an instructive contrast to the relatively incremental approach proposed here.

2.1. Limitations of current ASCII packet diagrams usage

ASCII packet header diagrams are commonplace in the IETF standards documents for binary protocols. While there is no standard for how these diagrams should be formatted, they have a broadly similar structure, where the layout of a protocol data unit (PDU) or structure is given in an ASCII diagram, and a description list of the fields that it contains are given immediately below. An example of this format is given in Figure 1.

```
      The RESET_STREAM frame is as follows:

       0                   1                   2                   3
       0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
      +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
      |                        Stream ID (i)                        ...
      +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
      |  Application Error Code (16)  |
      +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
      |                        Final Size (i)                       ...
      +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+

      RESET_STREAM frames contain the following fields:

      Stream ID:  A variable-length integer encoding of the Stream ID of
         the stream being terminated.

      Application Protocol Error Code:  A 16-bit application protocol
         error code (see Section 20.1) which indicates why the stream is
         being closed.

      Final Size:  A variable-length integer indicating the final size
         of the stream by the RESET_STREAM sender, in unit of bytes.

     Figure 1: QUIC's RESET_STREAM frame format (from [QUIC-TRANSPORT])
```

However, these diagrams, and their accompanying descriptions, are formatted for human readers rather than for machine parsing. As a result, while there is broad consistency in how ASCII packet diagrams are formatted, there are a number of limitations that are prohibitive to machine parsing:

Inconsistent syntax: There are two classes of consistency that are required for parsability: internal consistency, within a document or diagram, and external consistency, across all documents. Given that ASCII packet diagrams are formatted for human readers, rather than for machine parsing, there is sufficient variability in how they are formatted that parsing is difficult.

```
         The format of the "Relay Source Port Option" is shown below:

       0                   1                   2                   3
       0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
      +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
      |    OPTION_RELAY_PORT    |         Option-Len                  |
      +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
      |    Downstream Source Port     |
      +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

Where:

Option-Code: OPTION_RELAY_PORT. 16-bit value, 135.

Option-Len: 16-bit value to be set to 2.

Downstream Source Port: 16-bit value. To be set by the IPv6 relay either to the downstream relay agent's UDP source port used for the UDP packet, or to zero if only the local relay agent uses the non-DHCP UDP port (not 547).

Figure 2: DHCPv6's Relay Source Port Option (from [RFC8357])

Figure 1 gives an example of internal inconsistency. Here, the ASCII diagram shows a field labelled "Application Error Code", while the accompanying description lists the field as "Application Protocol Error Code". The use of an abbreviated name is suitable for human readers, but makes parsing the structure difficult for machines. Figure 2 gives a further example, where the description lists a field "Option-Code" that does not appear in the ASCII diagram. In addition, the description list describes each field as being 16 bits in length, while the diagram shows the OPTION_RELAY_PORT as 13 bits, and Option-Len as 19 bits. Another example of this -- where the diagram and accompanying text disagree -- is in [RFC6958], where the packet header diagram showing the structure of the Burst/Gap Loss Metrics Report Block shows the Number of Bursts field as being 12 bits wide but the corresponding text describes it as 16 bits.

Comparing Figure 1 with Figure 2 exposes external inconsistency across documents. While the ASCII diagrams themselves are broadly similar, the text surrounding the diagrams is formatted differently. If machine parsing is to be made possible, then this text must be structured consistently.

Ambiguous constraints: The constraints that are enforced on a particular field are often described ambiguously, or in a way that cannot be parsed easily. In Figure 2, each of the three fields in the structure is constrained. The first two fields ("Option-Code" and "Option-Len") are to be set to constant values (note the inconsistency in how these constraints are expressed in the description). However, the third field ("Downstream Source Port") can take a value from a constrained set. This constraint is expressed in prose that can easily be parsed by humans, but not by machines.

Poor linking between sub-structures: Protocol data units and other structures are often comprised of sub-structures that are defined

elsewhere, either in the same document, or within another document. Chaining these structures together is essential for machine parsing: the parsing process for a protocol data unit is only fully expressed if all elements can be parsed.

Figure 1 highlights the difficulty that machine parsers have in chaining structures together. Two fields ("Stream ID" and "Final Size") are described as being encoded as variable-length integers; this is a structure described elsewhere in the same document. Structured text is required both alongside the definition of the containing structure and with the definition of the sub-structure, to allow a parser to link the two together.

2.2. Formal languages in standards documents

A small proportion of IETF standards documents contain structured and formal languages, including ABNF [RFC5234], ASN.1 [ASN1], C, CBOR [RFC7049], JSON, the TLS presentation language [RFC8446], YANG models [RFC7950], and XML. While this broad range of languages may be problematic for the development of tooling to parse specifications, these, and other, languages serve a range of different use cases. ABNF, for example, is typically used to specify text protocols, while ASN.1 is used to specify data structure serialisation. This document specifies a structured language for specifying the parsing of binary protocol data units.

3. Design Principles

The use of structures that are designed to support machine readability may potentially interfere with the existing ways in which protocol specifications are used and authored. To the extent that these existing uses are more important than machine readability, such interference must be minimised.

In this section, the broad design principles that underpin the format described by this document are given. However, these principles apply more generally to any approach that introduces structured and formal languages into standards documents.

It should be noted that these are design principles: they expose the trade-offs that are inherent within any given approach. Violating these principles is sometimes necessary and beneficial, and this document sets out the potential consequences of doing so.

The central tenet that underpins these design principles is a recognition that the standardisation process is not broken, and so does not need to be fixed. Failure to recognise this will likely lead to approaches that are incompatible with the standards process,

or that will see limited adoption. However, the standards process can be improved with appropriate approaches, as guided by the following broad design principles:

Most readers are human: Primarily, standards documents should be written for people, who require text and diagrams that they can understand. Structures that cannot be easily parsed by people should be avoided, and if included, should be clearly delineated from human-readable content.

Any approach that shifts this balance -- that is, that primarily targets machine readers -- is likely to be disruptive to the standardisation process, which relies upon discussion centered around documents written in prose.

Authorship tools are diverse: Authorship is a distributed process that involves a diverse set of tools and workflows. The introduction of machine-readable structures into specifications should not require that specific tools are used to produce standards documents, to ensure that disruption to existing workflows is minimised. This does not preclude the development of optional, supplementary tools that aid in the authoring machine-readable structures.

The immediate impact of requiring specific tooling is that adoption is likely to be limited. A long-term impact might be that authors whose workflows are incompatible might be alienated from the process.

Canonical specifications: As far as possible, machine-readable structures should not replicate the human readable specification of the protocol within the same document. Such structures should form part of a canonical specification of the protocol. Adding supplementary machine-readable structures, in parallel to the existing human readable text, is undesirable because it could create the potential for inconsistency.

As an example, program code that describes how a protocol data unit can be parsed might be provided as an appendix within a standards document. This code would provide a specification of the protocol that is separate to the prose description in the main body of the document. This has the undesirable effect of introducing the potential for the program code to specify behaviour that the prose-based specification does not, and vice-versa.

Expressiveness: Any approach should be expressive enough to capture the syntax and parsing process for the majority of binary

protocols. If a given language is not sufficiently expressive, then adoption is likely to be limited. At the limits of what can be expressed by the language, authors are likely to revert to defining the protocol in prose: this undermines the broad goal of using structured and formal languages. Equally, though, understandable specifications and ease of use are critical for adoption. A tool that is simple to use and addresses the most common use cases might be preferred to a complex tool that addresses all use cases.

Minimise required change: Any approach should require as few changes as possible to the way documents are formatted, authored, and published. Forcing adoption of a particular structured or formal language is incompatible with the IETF's standardisation process: there are very few components of standards documents that are non-optional.

4. Augmented ASCII Packet Header Diagrams

The design principles described in Section 3 can largely be met by the existing uses of ASCII packet header diagrams. These diagrams aid human readability, do not require new or specialised authorship tools, do not split the specification into multiple parts, can express most binary protocol features, and require no changes to the existing publication processes.

However, as discussed in Section 2.1 there are limitations to how ASCII diagrams are used that must be addressed if they are to be parsed by machine. In this section, an augmented ASCII packet header diagram format is described.

The concept is first illustrated by example. This is appropriate, given the visual nature of the language. In future drafts, these examples will be parsable using provided tools, and a formal specification of the augmented ASCII packet diagrams will be given in Appendix A.

In the augmented ASCII packet diagrams, each protocol data unit is described in its own section of the document. This enables cross-referencing between data units using section numbering. In this specification-by-example, each element of the format will be described as part of a separate PDU.

4.1. Fixed-width Field Format

The simplest PDU is one that contains only a set of fixed-width fields in a known order, with no optional fields or variation in the packet format.

A Fixed-width Field Format packet is formatted as follows:

```
    0                   1                   2                   3
    0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
   +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
   |F2 |                         Field30                           |
   +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
   |                                                               |
   +                           Field64                             +
   |                                                               |
   +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
   |                                                               |
   +            Field48            +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
   |                               |
   +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
   |     Field8    |
   +-+-+-+-+-+-+-+-+
```

where:

Field2 (F2): 2 bits	This is a short field, and the diagram cannot show its full label. A short label (F2) is used in the diagram, and this short label is provided, in brackets, after the full label in the description list. The field's width -- 2 bits -- is given in the label of the description list, separated from the field's label by a colon.
Field30: 30 bits	This is a longer field whose full label can be shown in the diagram.
Field64: 8 bytes	This is a field that spans multiple rows. Where fields are an integral number of bytes in size, and start and end on a byte boundary, the field length can be given in bytes rather than in bits.
Field48: 48 bits	This is another multi-row field. As illustrated, fields are not required to end of a 32-bit boundary.
Field8: 1 byte	This field has been drawn on the next line, where it would have fit on the previous line. Where possible, the formatting of the diagram should be flexible to meet the needs of human readers.

4.2. Variable-width Field Format

Some packet formats include variable-width fields, where the size of a field is either derived from the value of some previous field, or is unspecified and inferred from the total size of the packet and the size of the other fields. A packet can contain only one unspecified length field, to ensure there is no ambiguity.

A Variable-width Field Format packet is formatted as follows:

```
     0                   1                   2                   3
     0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
    +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
    |     Field8    |
    +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
    |                      FieldVar - single row                 ...
    +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
    |                                                               :
    :                      FieldVar - multi-row                     :
    :                                                               |
    +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
    |                                                               :
    :              FieldVar - multi-row, unspecified length         :
    :                                                               |
    +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

where:

Field8 (F): 8 bits This is a fixed-width field, as described previously. As shown, while this field has a short label (F), this does not need to be used in the diagram.

FieldVar - single row: 2^F bits This is a variable-length field, whose length is defined by the value of the field with short label F (Field8). Constraint expressions can be used in place of constant values: the grammar for the expression language is defined in Section a.1. Where fields labels are used in a constraint, the field being referred to must have been defined before its label is used. Short variable-length fields are indicated by "..." instead of a pipe at the end of the row.

FieldVar - multi-row: 2^F bits This is a multi-row variable-length field, again constrained by the value of field F. Instead of the "..." notation, ":" is used to indicate that the field is variable-length. The use of ":" instead of "..." indicates the field is likely to be a longer, multi-row field. However, semantically, there is no difference: these different notations are for the benefit of human readers.

FieldVar - multi-row, unspecified length This is a variable-width field whose length is implied by the lengths of the other fields. At parsing time, the length of the PDU is known, and this can be used to determine the length of fields whose length is undefined. Each PDU can only leave a single field's length undefined: all other fields must be fixed-length, or have their widths constrained.

4.3. Cross-referencing and Sequences Format

A Cross-referencing and Sequences Format packet is formatted as follows:

```
     0                   1                   2                   3
     0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
    +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
    |     Field8    |
    +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
    |                                                               |
    +                                                               +
    |                                                               |
    +                        FieldFixedXRef                         +
    |                                                               |
    +                                                               +
    |                                                               |
    +                                               +-+-+-+-+-+-+-+-+
    |                                               |               :
    +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+               :
    :                                                               :
    :                          FieldVarXref                         :
    :                                                               |
    +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
    |                                                               |
    +                                                               +
    |                                                               |
    +                        [SeqFieldFixedXRef]                    +
    |                                                               |
    +                                                               +
    |                                                               |
    +                                               +-+-+-+-+-+-+-+-+
    |                                               |
    +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

where:

Field8 (F): 8 bits This is a fixed-width field, as described previously.

FieldFixedXRef: 1 Fixed-width Field Format This is a field whose structure is a previously defined PDU format. To indicate this, the width of the field is given in units of the cross-referenced structure (here, Fixed-width Field Format).

FieldVarXref: 1 Variable-width Field Format This field references a variable-width structure. It can be drawn to any width as appropriate, but must use a variable-width notation. Where multiple variable-width field format structures are referenced, the requirement that only one field's length can be unspecified applies to the enclosing structure.

SeqFieldFixedXRef: 2 Fixed-width Field Format Where a field is comprised of a sequence of previously defined structures, square brackets can be used to indicate this in the diagram. The length of the sequence can be defined using the constraint expression grammar as described earlier.

4.4. Optional Field Format

An Optional Field Format packet is formatted as follows:

```
     0                   1                   2                   3
     0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
    +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
    |     Field8    |
    +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
    |                          OptionalField                        |
    +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

where:

Field8 (F): 8 bits	This is a fixed-width field, as described previously.
OptionalField: 4 bytes	Present only when F > 3. This is a field whose presence is predicated on an expression given in the constraint expression grammar described earlier. Optional fields can be of any previously defined format (e.g., fixed- or variable-width). Optional fields are indicated by the presence of a "Present only when [expr]." as the first line in their description

5. IANA Considerations

This document contains no actions for IANA.

6. Security Considerations

Poorly implemented parsers are a frequent source of security vulnerabilities in protocol implementations. Structuring the description of a protocol data unit so that a parser can be automatically derived from the specification can reduce the likelihood of vulnerable implementations.

7. Acknowledgements

The authors would like to thank David Southgate for preparing a prototype implementation of some of the ideas described here.

This work has received funding from the UK Engineering and Physical Sciences Research Council under grant EP/R04144X/1.

8. Informative References

[ASN1] ITU-T, "ITU-T Recommendation X.680, X.681, X.682, and X.683", ITU-T Recommendation X.680, X.681, X.682, and X.683.

[QUIC-TRANSPORT] Iyengar, J. and M. Thomson, "QUIC: A UDP-Based Multiplexed and Secure Transport", draft-ietf-quic-transport-20 (work in progress), 23 April 2019, <http://www.ietf.org/internet-drafts/draft-ietf-quic-transport-20.txt>.

[RFC5234] Crocker, D. and P. Overell, "Augmented BNF for Syntax Specifications: ABNF", RFC 5234, January 2008, <https://www.rfc-editor.org/info/rfc5234>.

[RFC6958] Clark, A., Zhang, S., Zhao, J., and Q. Wu, "RTP Control Protocol (RTCP) Extended Report (XR) Block for Burst/Gap Loss Metric Reporting", RFC 6958, May 2013, <https://www.rfc-editor.org/info/rfc6958>.

[RFC7049] Bormann, C. and P. Hoffman, "Concise Binary Object Representation (CBOR)", RFC 7049, October 2013, <https://www.rfc-editor.org/info/rfc7049>.

[RFC7950] Bjorklund, M., "The YANG 1.1 Data Modeling Language",

              RFC 7950, August 2016,
              <https://www.rfc-editor.org/info/rfc7950>.

   [RFC8357] Deering, S. and R. Hinden, "Generalized UDP Source Port
              for DHCP Relay", RFC 8357, March 2018,
              <https://www.rfc-editor.org/info/rfc8357>.

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

Appendix A. ABNF specification

A.1. Constraint Expressions

```
       cond-expr = eq-expr "?" cond-expr ":" eq-expr
       eq-expr   = bool-expr eq-op   bool-expr
       bool-expr = ord-expr  bool-op ord-expr
       ord-expr  = add-expr  ord-op  add-expr

       add-expr  = mul-expr  add-op  mul-expr
       mul-expr  = expr      mul-op  expr
       expr      = *DIGIT / field-name

       field-name = *ALPHA

       mul-op  = "*" / "/" / "%"
       add-op  = "+" / "-"
       ord-op  = "<=" / "<" / ">=" / ">"
       bool-op = "&&" / "||" / "!"
       eq-op   = "==" / "!="
```

A.2. Augmented ASCII diagrams

Future revisions of this draft will include an ABNF specification for the augmented ASCII diagram format described in Section 4. Such a specification is omitted from this draft given that the format is likely to change as its syntax is developed. Given the visual nature of the format, it is more appropriate for discussion to focus on the examples given in Section 4.

Authors' Addresses

Stephen McQuistin
University of Glasgow
School of Computing Science
Glasgow
G12 8QQ

United Kingdom

Email: sm@smcquistin.uk

Vivian Band
University of Glasgow
School of Computing Science
Glasgow
G12 8QQ
United Kingdom

Email: vivianband0@gmail.com

Colin Perkins
University of Glasgow
School of Computing Science
Glasgow
G12 8QQ
United Kingdom

Email: csp@csperkins.org