Describing Protocol Data Units with Augmented Packet Header Diagrams
            draft-mcquistin-augmented-ascii-diagrams-03

Abstract

   This document describes a machine-readable format for specifying the
   syntax of protocol data units within a protocol specification. This
   format is comprised of a consistently formatted packet header
   diagram, followed by structured explanatory text. It is designed to
   maintain human readability while enabling support for automated
   parser generation from the specification document. This document is
   itself an example of how the format can be used.

Status of This Memo

   This Internet-Draft is submitted in full conformance with the
   provisions of BCP 78 and BCP 79.

   Internet-Drafts are working documents of the Internet Engineering
   Task Force (IETF). Note that other groups may also distribute
   working documents as Internet-Drafts. The list of current Internet-
   Drafts is at https://datatracker.ietf.org/drafts/current/.

   Internet-Drafts are draft documents valid for a maximum of six months
   and may be updated, replaced, or obsoleted by other documents at any
   time. It is inappropriate to use Internet-Drafts as reference
   material or to cite them other than as "work in progress."

   This Internet-Draft will expire on 10 September 2020.

Copyright Notice

Table of Contents

1. Introduction

   Packet header diagrams have become a widely used 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.

   Figure 1 gives an example of how packet header diagrams are used to
   define binary protocol formats. The format has an obvious structure:
   the diagram clearly delineates each field, showing its width and its
   position within the header. This type of diagram is designed for
   human readers, but is consistent enough that it should be possible to
   develop a tool that generates a parser for the packet format from the
   diagram.

```
:     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
:    +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
:    |          Source Port          |       Destination Port        |
:    +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
:    |                        Sequence Number                        |
:    +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
:    |                    Acknowledgment Number                      |
:    +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
:    |  Data |           |U|A|P|R|S|F|                               |
:    | Offset| Reserved  |R|C|S|S|Y|I|            Window             |
:    |       |           |G|K|H|T|N|N|                               |
:    +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
:    |           Checksum            |         Urgent Pointer        |
:    +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
:    |                    Options                    |    Padding    |
:    +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
:    |                             data                              |
:    +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```
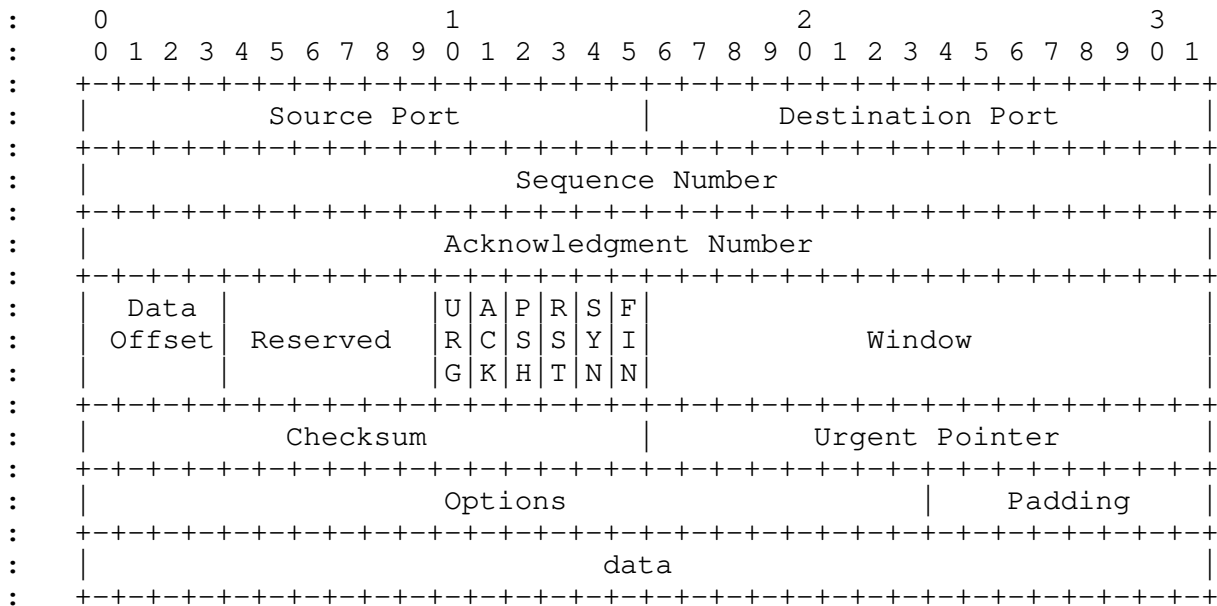
Figure 1: TCP's header format (from [RFC793])

Unfortunately, the format of such packet diagrams varies both within
and between documents. This variation makes it difficult to build
tools to generate parsers from the specifications. Better tooling
could be developed if protocol specifications adopted a consistent
format for their packet descriptions. Indeed, this underpins the
format described by this draft: we want to retain the benefits that
packet header diagrams provide, while identifying the benefits of
adopting a consistent format.

This document describes a consistent packet header diagram 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 writing, are described, before the format itself
is given.

This document is itself an example of the approach that it describes,
with the packet header diagrams and structured text format described
by example. Examples that do not form part of the protocol
description language are marked by a colon at the beginning of each
line; this prevents them from being parsed by the accompanying
tooling.

This draft describes early work. As consensus builds around the
particular syntax of the format described, both a formal ABNF

specification (Appendix A) and code (Appendix B) that parses it (and,
as described above, this document) will be provided.

2. Background

This section begins by considering how 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
format 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 Packet Format Diagrams

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

Packet header diagrams are frequently used in IETF standards to
describe the format of 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 shown in diagrammatic form, followed by a description
list of the fields that it contains. An example of this format,
taken from the QUIC specification, is given in Figure 2.

These packet header diagrams, and the accompanying descriptions, are
formatted for human readers rather than for automated processing. As
a result, while there is rough consistency in how packet header
diagrams are formatted, there are a number of limitations that make
them difficult to work with programmatically:

Inconsistent syntax: There are two classes of consistency that are
   needed to support automated processing of specifications: internal
   consistency within a diagram or document, and external consistency
   across all documents.

   Figure 2 gives an example of internal inconsistency. Here, the
   packet 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 3 gives a further example, where the description
   includes an "Option-Code" field that does not appear in the packet
   diagram; and where the description states that each field is 16
   bits in length, but the diagram shows the OPTION_RELAY_PORT as 13
   bits, and Option-Len as 19 bits. Another example is [RFC6958],
   where the packet format 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 2 with Figure 3 exposes external inconsistency
   across documents. While the packet format diagrams are broadly
   similar, the surrounding text 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 3, 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 cannot readily by understood by machine.

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 2 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.

Lack of extension and evolution syntax: Protocols are often
   specified across multiple documents, either because the protocol
   explicitly includes extension points (e.g., profiles and payload
   format specifications in RTP [RFC3550]) or because definition of a
   protocol data unit has changed and evolved over time. As a
   result, it is essential that syntax be provided to allow for a
   complete definition of a protocol's parsing process to be
   constructed across multiple documents.
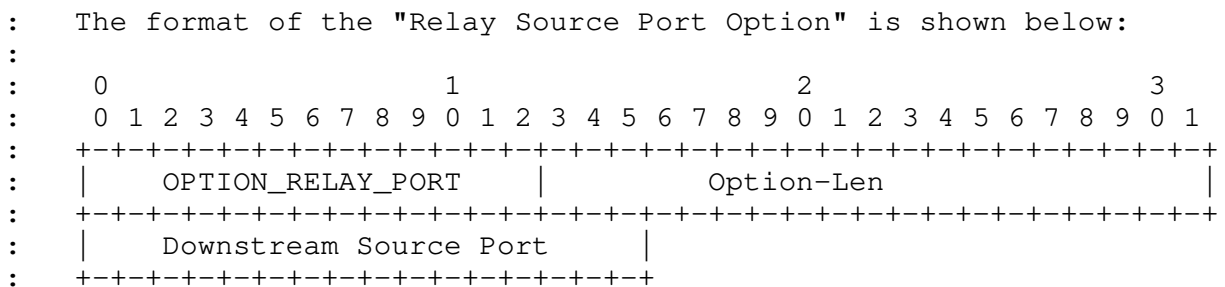
```
:   The format of the "Relay Source Port Option" is shown below:
:
:    0                   1                   2                   3
:    0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
:   +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
:   |      OPTION_RELAY_PORT    |           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 3: DHCPv6's Relay Source Port Option (from [RFC8357])

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 might 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.

   Writing tools are diverse: Standards document writing 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. Machine-readable
      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 creates 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.

      It may be desirable to restrict expressiveness, however, to
      guarantee intrinsic safety, security, and computability properties

of both the generated parser code for the protocol, and the parser
of the description language itself. In much the same way as the
language-theoretic security ([LANGSEC]) community advocates for
programming language design to be informed by the desired
properties of the parsers for those languages, protocol designers
should be aware of the implications of their design choices. The
expressiveness of the protocol description languages that they use
to define their protocols can force such awareness.

Broadly, those languages that have grammars which are more
expressive tend to have parsers that are more complex and less
safe. As a result, while considering the other goals described in
this document, protocol description languages should attempt to be
minimally expressive, and either restrict protocol designs to
those for which safe and secure parsers can be generated, or as a
minimum, ensure that protocol designers are aware of the
boundaries their designs cross, in terms of computability and
decidability [SASSAMAN].

Minimise required change: Any approach should require as few changes
as possible to the way that 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 Packet Header Diagrams

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

However, as discussed in Section 2.1 there are limitations to how
packet header diagrams are used that must be addressed if they are to
be parsed by machine. In this section, an augmented 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 packet diagrams will be given in
Appendix A.

4.1. PDUs with Fixed and Variable-Width Fields

   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.

   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.

   To ensure that there is no ambiguity, a PDU description can contain
   only one field whose length is unspecified. The length of a single
   field, where all other fields are of known (but perhaps variable)
   length, can be inferred from the total size of the containing PDU.

   A PDU description is introduced by the exact phrase "A/An _____ is
   formatted as follows:" at the end of a paragraph. This is followed
   by the PDU description itself, as a packet diagram within an
   <artwork> element in the XML representation, starting with a header
   line to show the bit width of the diagram. The description of the
   fields follows the diagram, as an XML <dl> list, after a paragraph
   containing the text "where:".

   PDU names must be unique, both within a document, and across all
   documents that are linked together (i.e., using the structured
   language defined in Section 4.4).

   Each field of the description starts with a <dt> tag comprising the
   field name and an optional short name in parenthesis. These are
   followed by a colon, the field length, an optional presence
   expression (described in Section 4.2), and a terminating period. The
   following <dd> tag contains a prose description of the field. Field
   names cannot be the same as a previously defined PDU name, and must
   be unique within a given structure definition.

   For example, this can be illustrated using the IPv4 Header Format
   [RFC791]. An IPv4 Header 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
         +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
         |Version|  IHL  |    DSCP   |ECN|          Total Length         |
         +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
         |         Identification        |Flags|      Fragment Offset    |
         +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
         |  Time to Live |    Protocol   |         Header Checksum        |
         +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
         |                         Source Address                        |
         +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
         |                      Destination Address                      |
         +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
         |                            Options                         ...
         +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
         |                                                               :
         :                            Payload                           :
         :                                                               |
         +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

   where:

   Version (V): 4 bits. This is a fixed-width field, whose full label
      is shown in the diagram. The field's width -- 4 bits -- is given
      in the label of the description list, separated from the field's
      label by a colon.

   Internet Header Length (IHL): 4 bits. This is a shorter field, whose
      full label is too large to be shown in the diagram. A short label
      (IHL) is used in the diagram, and this short label is provided, in
      brackets, after the full label in the description list.

   Differentiated Services Code Point (DSCP): 6 bits. This is a fixed-
      width field, as previously discussed.

   Explicit Congestion Notification (ECN): 2 bits. This is a fixed-
      width field, as previously discussed.

   Total Length (TL): 2 bytes. This is a fixed-width field, as
      previously discussed. Where fields are an integral number of
      bytes in size, the field length can be given in bytes rather than
      in bits.

   Identification: 2 bytes. This is a fixed-width field, as previously
      discussed.

   Flags: 3 bits. This is a fixed-width field, as previously discussed.

   Fragment Offset: 13 bits. This is a fixed-width field, as previously
      discussed.

   Time to Live (TTL): 1 byte. This is a fixed-width field, as
      previously discussed.

   Protocol: 1 byte. This is a fixed-width field, as previously
      discussed.

   Header Checksum: 2 bytes. This is a fixed-width field, as previously
      discussed.

   Source Address: 32 bits. This is a fixed-width field, as previously
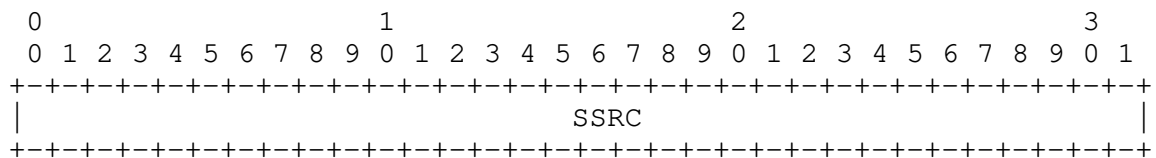   discussed.

   Destination Address: 32 bits. This is a fixed-width field, as
      previously discussed.

   Options: (IHL-5)*32 bits. This is a variable-length field, whose
      length is defined by the value of the field with short label IHL
      (Internet Header Length). Constraint expressions can be used in
      place of constant values: the grammar for the expression language
      is defined in Appendix A.1. Constraints can include a previously
      defined field's short or full label, where one has been defined.
      Short variable-length fields are indicated by "..." instead of a
      pipe at the end of the row.

   Payload: TL - ((IHL*32)/8) bytes. This is a multi-row variable-
      length field, constrained by the values of fields TL and IHL.
      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.

4.2. PDUs That Cross-Reference Previously Defined Fields

   Binary formats often reference sub-structures that have been defined
   earlier in the specification. For example, in RTP [RFC3550], the
   Contributing Source Identifiers in an RTP Data Packet are defined as
   comprising a list of Source Identifier elements. A Source Identifier
   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
   +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
   |                             SSRC                              |
   +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```
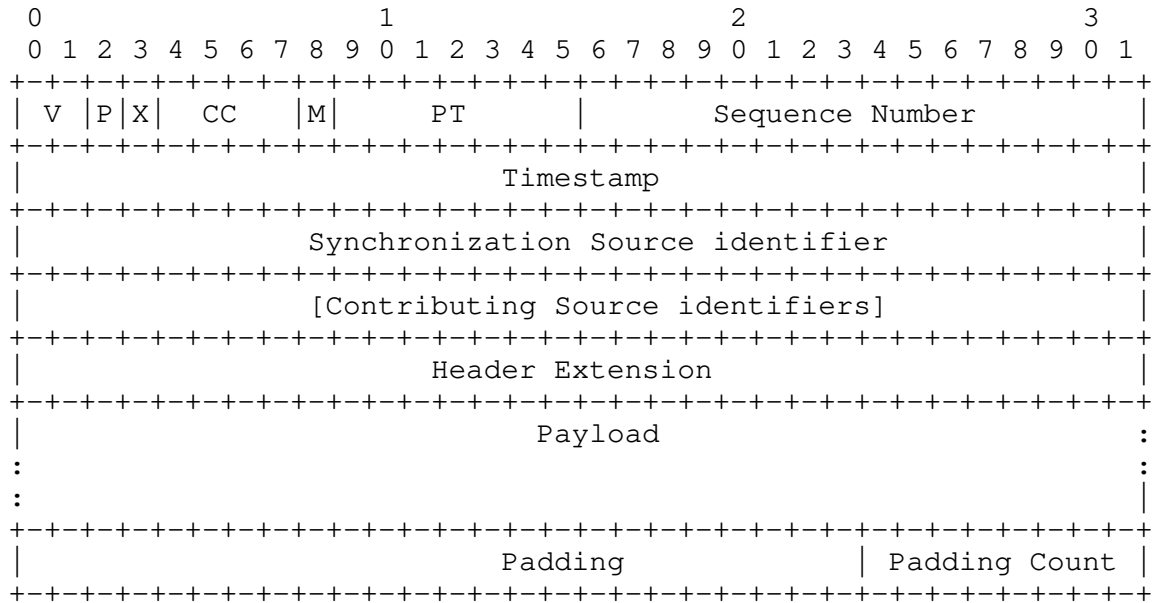
   where:

   SSRC: 32 bits. This is a fixed-width field, as described previously.

   The following example shows how a Source Identifier can be referenced
   in the description of an RTP Data Packet. It also shows how the
   presence of some fields in a format may be dependent on the values of
   an earlier field.

   An RTP Data 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
      +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
      | V |P|X|  CC   |M|     PT      |        Sequence Number         |
      +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
      |                           Timestamp                           |
      +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
      |                 Synchronization Source identifier             |
      +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
      |                 [Contributing Source identifiers]             |
      +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
      |                        Header Extension                       |
      +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
      |                           Payload                             :
      :                                                               :
      :                                                               |
      +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
      |                  Padding                    | Padding Count  |
      +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

   where:

   Version (V): 2 bits. This is a fixed-width field, as described
      previously.

   Padding (P): 1 bit. This is a fixed-width field, as described
      previously.

   Extension (X): 1 bit. This is a fixed-width field, as described
      previously.

   CSRC count (CC): 4 bits. This is a fixed-width field, as described
      previously.

   Marker (M): 1 bit. This is a fixed-width field, as described
      previously.

   Payload Type (PT): 7 bits. This is a fixed-width field, as described
      previously.

   Sequence Number (PT): 16 bits. This is a fixed-width field, as
      described previously.

   Timestamp (PT): 32 bits. This is a fixed-width field, as described
      previously.

   Synchronization Source identifier: 1 * Source Identifier. This is a
      field whose structure is a previously defined PDU format (Source
      Identifier). To indicate this, the width of the field is
      expressed in terms of cross-referenced structure. When used in
      constraint expressions, PDU names refer to the length of that PDU
      structure.

   Contributing Source identifiers: CC * Source Identifier. 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.

      In this example, both a PDU name (Source Identifier) and a field
      name (CC) are used in the constraint expression. The PDU name
      refers to the length of the PDU, while the field name refers to
      the value of the field. This is possible because field names
      cannot be the same as previously defined PDU names.

   Header Extension: 32 bits; present only when X == 1. This is a field
      whose presence is predicated on an expression given using 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 ";
      present only when [expr]." at the end of the definition term
      (i.e., the text contained within the <dt> tag).

      [Note that this example deviates from the format as described in
      [RFC3550]. As specified in that document, the Header Extension
      would be a cross-referenced structure. This is not shown here for
      brevity.]

   Payload. The length of the Payload is not specified, and hence needs
      to be inferred from the total length of the packet and the lengths
      of the known fields. There can only be one field of unspecified
      size in a PDU.

   Padding: Padding Count bytes; present only when (P == 1) and
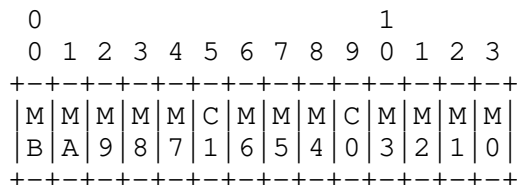   (Padding Count > 0).

This is a variable size field, with size dependent on a later
field in the packet. Fields can only depend on the value of a
later field if they follow a field with unspecified size.

Padding Count: 1 byte; present only when P == 1. This is a fixed-
width field, as previously discussed.

4.3. PDUs with Non-Contiguous Fields

In some binary formats, fields are striped across multiple non-
contiguous bits. This is often to allow for backwards compatibility
with previous definitions of the same fields in earlier documents:
striping in this way allows for careful use of the possible range of
values.

This format is illustrated using the STUN Message Type
[draft-ietf-tram-stunbis-21]. A STUN Message Type is formatted as
follows:

```
 0                   1
 0 1 2 3 4 5 6 7 8 9 0 1 2 3
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|M|M|M|M|M|C|M|M|M|C|M|M|M|M|
|B|A|9|8|7|1|6|5|4|0|3|2|1|0|
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

where:

Method (M): 12 bits. This field is comprised of multiple sub-fields
   (M0 through MB) as shown in the diagram. That these sub-fields
   should be concatenated, after parsing, into a single field is
   indicated by their being labelled using the 'M' short field name
   followed by a single hexadecimal digit, with the least significant
   bit labelled with 0, and subsequent bits labelled in sequence.

Class (C): 2 bits. This field follows the same format as M described
   above.

4.4. Importing PDU Definitions from Other Documents

Protocols are often specified across multiple documents, either
because the specification of a protocol's data units has changed over
time, or because of explicit extension points contained in the
protocol's original specification. To allow a document to make use
of a previous PDU definition, it is possible to import PDU
definitions (written in the format described in this document) from
other documents.

A PDU definition is imported using the exact phrase "A/An _____ is
formatted as described in <document identifier>". The document
identifier must refer, unambiguously, to an existing document. An
Internet-Draft is identified by its name. RFCs are identified by
"RFC" followed by their number.

5. Open Issues

   * Need a simple syntax for defining a list of identical objects, and
      a way of referring to the size of the enclosing packet. The
      format cannot currently represent RFC 6716 section 3.2.3, and
      should be able to (the underlying type system can do so).

   * Need some discussion about the checks that the tooling might
      perform, and the implications of those checks. For example, the
      tooling checks for consistency between the diagram and the
      description list of fields, ensuring that fields match by name and
      width. -01 of this draft had a field that mismatched because of
      case: is this something that the tooling should identify? More
      broadly, what is the trade-off between the rigour that the tooling
      can enforce, and the flexibility desired/needed by authors?

   * Need to describe the rules governing the import of PDU definitions
      from other documents.

6. IANA Considerations

   This document contains no actions for IANA.

7. 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.

   As described in Section 3, the expressiveness of a protocol
   description language has implications for the safety, security, and
   computability properties of the parser for the protocol description
   language itself, and on the generated parser code for the protocols
   described using it. The language-theoretic security ([LANGSEC])
   community explores the security implications of programming language
   design; the principles developed in that community should guide the
   development of protocol description languages.

8. Acknowledgements

9. Informative References

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

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

   [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>.

   [RFC7950] Bjorklund, M., "The YANG 1.1 Data Modeling Language",
              RFC 7950, August 2016,
              <https://www.rfc-editor.org/info/rfc7950>.

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

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

   [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.

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

   [RFC3550]   Schulzrinne, H., Casner, S., Frederick, R., and V.
               Jacobson, "RTP: A Transport Protocol for Real-Time
               Applications", RFC 3550, July 2003,
               <https://www.rfc-editor.org/info/rfc3550>.

   [draft-ietf-tram-stunbis-21]
               Petit-Huguenin, M., Salgueiro, G., Rosenberg, J., Wing,
               D., Mahy, R., and P. Matthews, "Session Traversal
               Utilities for NAT (STUN)", Work in Progress, Internet-
               Draft, draft-ietf-tram-stunbis-21, 21 March 2019,
               <http://www.ietf.org/internet-drafts/draft-ietf-tram-
               stunbis-21.txt>.

   [RFC791]   Postel, J., "Internet Protocol", RFC 791, September 1981,
               <https://www.rfc-editor.org/info/rfc791>.

   [RFC793]   Postel, J., "Transmission Control Protocol", RFC 793,
               September 1981, <https://www.rfc-editor.org/info/rfc793>.

   [LANGSEC]   LANGSEC, "LANGSEC: Language-theoretic Security",
               <http://langsec.org>.

   [SASSAMAN] Sassaman, L., Patterson, M. L., Bratus, S., and A.
               Shubina, "The Halting Problems of Network Stack
               Insecurity", ;login: -- December 2011, Volume 36, Number
               6, <https://www.usenix.org/publications/login/december-
               2011-volume-36-number-6/halting-problems-network-stack-
               insecurity>.

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-ws / "(" expr ")"

      field-name    = *ALPHA
      field-name-ws = *(field-name " ")

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

A.2. Augmented packet diagrams

   Future revisions of this draft will include an ABNF specification for
   the augmented packet 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.

Appendix B. Source code repository

   The source for this draft is available from https://github.com/
   glasgow-ipl/draft-mcquistin-augmented-ascii-diagrams.

   The source code for tooling that can be used to parse this document
   is available from https://github.com/glasgow-ipl/ips-protodesc-code.

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

Dejice Jacob
University of Glasgow
School of Computing Science
Glasgow
G12 8QQ
United Kingdom

Email: d.jacob.1@research.gla.ac.uk

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

Email: csp@csperkins.org