Parsing Protocol Standards to Parse Standard Protocols

Stephen McQuistin
Vivian Band
Dejice Jacob
Colin Perkins

NGN Seminar, Friday 6th November 2020

This work is funded by the UK Engineering and Physical Sciences Research Council, under grant EP/R04144X/1.
• Internet standards documents are typically written in English prose

• As protocols become more complex, this becomes undesirable

• Inconsistencies and ambiguities are easily introduced by natural language descriptions

• Formal specification languages would make documents more concise and consistent, and enable machine parsing
Internet standards documents are typically written in English prose. As protocols become more complex, this becomes undesirable. Inconsistencies and ambiguities are easily introduced by natural language descriptions. Formal specification languages would make documents more concise and consistent, and enable machine parsing.
… to Parse Standard Protocols

- Machine readability would enable automatic code generation
- This enables testing of the protocol specification as it develops
- Modern, secure systems languages can be supported
- Overall, the security and trustworthiness of standards may be improved
Augmented BNF for Syntax Specifications: ABNF

Status of This Memo
This document specifies an Internet standards track protocol for the Internet community, and requests discussion and suggestions for improvements. Please refer to the current edition of the “Internet Official Protocol Standards” (STD 1) for the standardization state and status of this protocol. Distribution of this memo is unlimited.

Abstract
Internet technical specifications often need to define a formal syntax. Over the years, a modified version of Backus-Naur Form (BNF), called Augmented BNF (ABNF), has been popular among many Internet specifications. The current specification documents ABNF. It balances compactness and simplicity with reasonable representational power. The differences between standard BNF and ABNF involve naming rules, repetition, alternatives, order-independence, and value ranges. This specification also supplies additional rule definitions and encoding for a core lexical analyzer of the type common to several Internet specifications.
Augmented BNF for Syntax Specifications: ABNF

Abstract

Internet technical specifications often need to define a formal syntax. Over the years, a modified version of Backus-Naur Form (BNF), called Augmented BNF (ABNF), has been popular among many Internet specifications. The current specification documents ABNF. It balances compactness and simplicity with reasonable representational power. The differences between standard BNF and ABNF involve naming rules, repetition, alternatives, order-independence, and value ranges. This specification also supplies additional rule definitions and encoding for a core lexical analyzer of the type common to several Internet specifications.

 Crocker & Overell           Standards Track                     [Page 1]
 Network Working Group                                       S. McQuistin
 Internet-Draft                                                   V. Band
 Intended status: Experimental                                   D. Jacob
 Expires: 19 December 2020                                  C. S. Perkins
 University of Glasgow
 17 June 2020
 Describing QUIC's Protocol Data Units with Augmented Packet Header
 Diagams
draft-mcquistin-quic-augmented-diagrams-01

Abstract

This document describes the core transport protocol data units used in the QUIC protocol using a machine-readable augmented packet header diagram format. It is intended as an example of the diagram format, and not as a contribution to the development of the QUIC protocol.

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 19 December 2020.

Copyright Notice

Copyright (c) 2020 IETF Trust and the persons identified as the document authors. All rights reserved.

Augmented BNF for Syntax Specifications: ABNF

Status of This Memo

This document specifies an Internet standards track protocol for the Internet community, and requests discussion and suggestions for improvements. Please refer to the current edition of the "Internet Official Protocol Standards" (STD 1) for the standardization state and status of this protocol. Distribution of this memo is unlimited.

Abstract

Internet technical specifications often need to define a formal syntax. Over the years, a modified version of Backus-Naur Form (BNF), called Augmented BNF (ABNF), has been popular among many Internet specifications. The current specification documents ABNF. It balances compactness and simplicity with reasonable representational power. The differences between standard BNF and ABNF involve naming rules, repetition, alternatives, order-independence, and value ranges. This specification also supplies additional rule definitions and encoding for a core lexical analyzer of the type common to several Internet specifications.

Describing QUIC's Protocol Data Units with Augmented Packet Header Diagrams
draft-mcquistin-quic-augmented-diagrams-01

Abstract

This document describes the core transport protocol data units used in the QUIC protocol using a machine-readable augmented packet header diagram format. It is intended as an example of the packet header diagram language, and not as a contribution to the development of the QUIC protocol.

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 19 December 2020.

Copyright Notice

Copyright (c) 2020 IETF Trust and the persons identified as the document authors. All rights reserved.
Abstract

Internet technical specifications often need to define a formal syntax. Over the years, a modified version of Backus-Naur Form (BNF), called Augmented BNF (ABNF), has been popular among many Internet specifications. The current specification documents ABNF. It balances compactness and simplicity with reasonable representational power. The differences between standard BNF and ABNF involve naming rules, repetition, alternatives, order-independence, and value ranges. This specification also supplies additional rule definitions and encoding for a core lexical analyzer of the type common to several Internet specifications.
What are the requirements of a common protocol representation?
Representing Protocol Data

• Syntax description languages
  • ABNF, ASN.1, the TLS 1.3 presentation language, …
Representing Protocol Data

- Syntax description languages
  - ABNF, ASN.1, the TLS 1.3 presentation language, …

These languages can only be used to describe protocol syntax
Representing Protocol Data

- **Syntax description languages**
  - ABNF, ASN.1, the TLS 1.3 presentation language, …

- **Protocol type systems**
  - eTPL, YANG, NetPDL, PADS, DataScript, PacketTypes, the Meta Packet Language, …
Representing Protocol Data

• **Syntax description languages**
  - ABNF, ASN.1, the TLS 1.3 presentation language, …

• **Protocol type systems**
  - eTPL, YANG, NetPDL, PADS, DataScript, PacketTypes, the Meta Packet Language, …

These languages couple external and internal representations: can’t model protocols where these are different
Representing Protocol Data

• **Syntax description languages**
  • ABNF, ASN.1, the TLS 1.3 presentation language, …

• **Protocol type systems**
  • eTPL, YANG, NetPDL, PADS, DataScript, PacketTypes, the Meta Packet Language, …

• **Protocol representation systems**
  • Nail, Narcissus, …
Representing Protocol Data

- **Syntax description languages**
  - ABNF, ASN.1, the TLS 1.3 presentation language, …

- **Protocol type systems**
  - eTPL, YANG, NetPDL, PADS, DataScript, PacketTypes, the Meta Packet Language, …

- **Protocol representation systems**
  - Nail, Narcissus,

Need support for strong type guarantees *and* support for context-based, multi-stage parsing
We need a common representation that is safe and extensible
The Network Packet Representation

- A typed protocol representation
- Decoupled from protocol description languages and target output languages
- Provides type constructors for a number of basic type classes, that can be composed into descriptions for complex protocols
With these requirements in mind, Section 4 describes the within a wider architecture, being generated by an input within and across di...

The Network Packet Representation

An RTP Data Packet is formatted as follows:

```
<table>
<thead>
<tr>
<th>V=2</th>
<th>P</th>
<th>X</th>
<th>CC</th>
<th>M</th>
<th>PT</th>
<th>sequence number</th>
</tr>
</thead>
<tbody>
<tr>
<td>timestamp</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>synchronization source (SSRC) identifier</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>[CSRC identifier list]</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>(4 * CC octets)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CC may be zero</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>defined by signalling</td>
<td>header extension length</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>header extension</td>
<td>OPTIONAL</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>format defined by signalling</td>
<td>(if X=1)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Payload</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>(variable format and length, depends on PT)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>+-----+---------------------+-------------</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Padding (PadCnt octets, if P=1)</td>
<td>PadCnt (if P=1)</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
```
The Network Packet Representation

An RTP Data Packet is formatted as follows:

<table>
<thead>
<tr>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>V=2</th>
<th>Bit</th>
<th>CC</th>
<th>M</th>
<th>PT</th>
<th>sequence number</th>
</tr>
</thead>
<tbody>
<tr>
<td>timestamp</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>synchronization source (SSRC) identifier</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>[SSRC Identifier list]</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>[header extension]</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>format defined by signalling</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>[Payload]</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>(variable format and length, depends on PT)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>[Padding (PadCnt octets, if P=1)</td>
<td>PadCnt (if P=1)]</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

where:

Version (V): 2 bits; V == 2. This field identifies the version of RTP. The version defined by this specification is two (2). (The value 1 is used by the first draft version of RTP and the value 0 is used by the protocol initially implemented in the "vat" audio tool.)

Padding (P): 1 bit. If the padding bit is set...

Bit strings to represent raw protocol data
The Network Packet Representation

An RTP Data Packet is formatted as follows:

<p>| +-----------------------------------------------+                              |</p>
<table>
<thead>
<tr>
<th>V</th>
<th>P</th>
<th>Pad Count (if P = 1)</th>
<th>Pad (if P = 1)</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td></td>
</tr>
<tr>
<td>0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>+-----------------------------------------------+</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Sequence number</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>+-----------------------------------------------+</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Timestamp</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>+-----------------------------------------------+</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Synchronization source (SSRC) identifier</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>+-----------------------------------------------+</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>[CSRC identifier list]</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>(4 * CC octets)</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CC may be zero</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>+-----------------------------------------------+</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Header extension length</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>+-----------------------------------------------+</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Header extension</td>
<td>OPTIONAL</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>+-----------------------------------------------+</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Format defined by signalling</td>
<td>(if X=1)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>+-----------------------------------------------+</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Payload</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>+-----------------------------------------------+</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

where:

Version (V): 2 bits; V = 2. This field identifies the version of RTP.
The version defined by this specification is two (2). (The value 1
is used by the first draft version of RTP and the value 0 is used by
the protocol initially implemented in the "vat" audio tool.)

Padding (P): 1 bit. If the padding bit is set,

Arrays to represent sequences of elements of the same type
The Network Packet Representation

An RTP Data Packet is formatted as follows:

| V=2|P|X | CC | | PT | sequence number |
|-----------------------------------------------|
| ++------------------------------------------------------------------|
| | timestamp |
| +------------------------------------------------------------------|
| | synchronization source (SSRC) identifier |
| +------------------------------------------------------------------|
| | [CSRC identifier list] |
| | (4 * CC octets) |
| +------------------------------------------------------------------|
| | CC may be zero |
| | header extension length |
| +------------------------------------------------------------------|
| | header extension |
| | format defined by signalling |
| +------------------------------------------------------------------|
| | Payload |
| | (variable format and length, depends on PT) |
| +------------------------------------------------------------------|
| | Padding (Padding octets, if P=1)|Padding (if P=1) |

where:

Version (V): 1 bit
The version defined by this specification is two (2). (The value 1 is used by the first draft version of RTP and the value 0 is used by documents format the diagrams and accompanying text. This lack of formalisation makes it difficult to parse these diagrams automatically.

Structures to represent packets themselves
An RTP Data Packet is formatted as follows:

```
+----------------+-----------------+-----------------+-----------------+-----------------+-----------------+-----------------+-----------------+-----------------+-----------------+-----------------+-----------------+-----------------+-----------------+-----------------+
| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 0 | 1 |
+----------------+-----------------+-----------------+-----------------+-----------------+-----------------+-----------------+-----------------+-----------------+-----------------+-----------------+-----------------+-----------------+-----------------+-----------------+-----------------+-----------------+-----------------+-----------------+-----------------+-----------------+-----------------+-----------------+-----------------+-----------------+
| V=2 | P[X] | CC | M | PT | | sequence number | | timestamp | | synchronization source (SSRC) identifier | [CSRC identifier list] | (4 * CC octets) | CC may be zero | defined by signalling | header extension length | || header extension | format defined by signalling | OPTIONAL | (if X=1) | |
+----------------+-----------------+-----------------+-----------------+-----------------+-----------------+-----------------+-----------------+-----------------+-----------------+-----------------+-----------------+-----------------+-----------------+-----------------+-----------------+-----------------+-----------------+-----------------+-----------------+-----------------+-----------------+-----------------+-----------------+-----------------+
| Payload | | (variable format and length, depends on PT) | |
+----------------+-----------------+-----------------+-----------------+-----------------+-----------------+-----------------+-----------------+-----------------+-----------------+-----------------+-----------------+-----------------+-----------------+-----------------+-----------------+-----------------+-----------------+-----------------+-----------------+-----------------+-----------------+-----------------+-----------------+-----------------+

where:

Version (V): 2 bits; V = 2. This field identifies the version of RTP. The version defined by this specification is two (2). (The value 1 is used by the first draft version of RTP and the value 0 is used by the protocol initially implemented in the "vat" audio tool.)

Padding (P): 1 bit. If the padding bit is set,

Constraints within structures

```
The Network Packet Representation

An RTP Data Packet is formatted as follows:

```
+---------------------------------------------+
| V=2 | P | X | CC | M | PT | sequence number |
| timestamp |
| synchronization source (SSRC) identifier |
| [CSRC identifier list] |
| (4 * CC octets) |
| CC may be zero |
| header extension length |
| defined by signalling |
| header extension |
| format defined by signalling |
| OPTIONAL |
| (if X=1) |
| Payload |
| (variable format and length, depends on PT) |
+---------------------------------------------+
```

Constraints within structures

where:

Version (V): 2 bits; V = 2. This field identifies the version of RTP. The version defined by this specification is two (2). (The value 1 is used by the first draft version of RTP and the value 0 is used by the protocol initially implemented in the "vat" audio tool.)

Padding (P): 1 bit. If the padding bit is set,
The Network Packet Representation

An RTP Data Packet is formatted as follows:

```
  +----------------+------------------+
  | Version (V)    | Payload Type (PT)|
  +----------------+------------------+
  |                  |                  |
  +----------------+------------------+
  |                  |                  |
  +----------------+------------------+
  |                  |                  |
  +----------------+------------------+
  |                  |                  |
  +----------------+------------------+
  |                  |                  |
  +----------------+------------------+
  |                  |                  |
  +----------------+------------------+
  |                  |                  |
  +----------------+------------------+
  |                  |                  |
  +----------------+------------------+
  |                  |                  |
  +----------------+------------------+
  |                  |                  |
  +----------------+------------------+
  |                  |                  |
  +----------------+------------------+
  |                  |                  |
  +----------------+------------------+
  | Padding (PadCnt octets, if P=1) PadCnt (if P=1) |
```

where:

- **Version (V):** 2 bits; \(V = 2\). This field identifies the version of RTP. The version defined by this specification is two (2). (The value 1 is used by the first draft version of RTP and the value 0 is used by the protocol initially implemented in the "vat" audio tool.)
- **Padding (P):** 1 bit. If the padding bit is set.
The Network Packet Representation

An RTP Data Packet is formatted as follows:

```
|V=2|P|X| CC |M| PT | sequence number |
+-----------------+-----------------+-----------------+-----------------+-----------------+-----------------+-----------------+-----------------+-----------------+
| padding (PadCnt octets, if P=1) | PadCnt (if P=1) |
```

where:

- Version (V): 2 bits; V = 2. This field identifies the version of RTP. The version defined by this specification is two (2). (The value 1 is used by the first draft version of RTP and the value 0 is used by the protocol initially implemented in the "vat" audio tool.)
- Padding (P): 1 bit. If the padding bit is set,
 Parsing Functions

• PDUs may have multi-stage parsing processes, with decryption or decompression necessary
Parsing Functions

- PDUs may have multi-stage parsing processes, with decryption or decompression necessary.

Bit String

Structure: Protected Packet

Structure: Unprotected Packet
• PDUs may have multi-stage parsing processes, with decryption or decompression necessary
Parsing Functions

- PDUs may have multi-stage parsing processes, with decryption or decompression necessary.
The Network Packet Representation

- A typed intermediate protocol representation, independent of input and output languages
- Enables state to be maintained between the parsing of different PDUs using typed parsing contexts
- Provides support for dependently formatted PDUs, constraints on and between PDU fields, and for multi-stage parsing via typed functions: all needed for parsing complex protocols
Network Packet Representation
There are social barriers to the adoption of protocol description techniques.
Integrating with Protocol Standards

- Most readers are human
- Authorship workflows are diverse
- Canonical specifications
- Expressiveness
- Minimise required change
Protocol Description Languages

• A wide number of languages are already in use: ABNF, ASN.1, YANG, the TLS 1.3 presentation language, …

• Any tool that aims to see broad adoption should accept multiple description formats

• The Network Packet Representation supports this: it is language agnostic

• Parsing structured description languages is well understood, and it should be possible to generate a Network Packet Representation from them

• Informal languages, like packet header diagrams, are more challenging
Augmented Packet Header Diagrams: QUIC example

An Initial Packet is formatted as follows:

```
0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+----------------+
| 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 |
+----------------+
| T | R | P | +----------------+
| Version +----------------+
| DCID Len +----------------+
| Destination Connection ID (DCID) ... +----------------+
| SCID Len +----------------+
| Source Connection ID (SCID) ... +----------------+
| Token Length ... +----------------+
| Token ... +----------------+
| Length ... +----------------+
| Packet Number ... +----------------+
| Payload ... +----------------+
```

where:

Header Form (HF): 1 bit; HF == 1. The most significant bit (0x80) of byte 0 (the first byte) is set to 1 for long header packets.
Augmented Packet Header Diagrams: QUIC example

An Initial Packet is formatted as follows:

<table>
<thead>
<tr>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
</tr>
</thead>
<tbody>
<tr>
<td>01234567890123456789012345678901</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>++----------------------------------</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>[1][1]T</td>
<td>R</td>
<td>P</td>
</tr>
<tr>
<td>++----------------------------------</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Version</td>
<td></td>
<td></td>
</tr>
<tr>
<td>++----------------------------------</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>DCID Len</td>
<td></td>
<td></td>
</tr>
<tr>
<td>++----------------------------------</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Destination Connection ID (DCID)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>++----------------------------------</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>SCID Len</td>
<td></td>
<td></td>
</tr>
<tr>
<td>++----------------------------------</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Source Connection ID (SCID)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>++----------------------------------</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Token Length</td>
<td></td>
<td></td>
</tr>
<tr>
<td>++----------------------------------</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Token</td>
<td></td>
<td></td>
</tr>
<tr>
<td>++----------------------------------</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Payload</td>
<td></td>
<td></td>
</tr>
<tr>
<td>++----------------------------------</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
| where:

Header Form (HF): 1 bit; HF == 1. The most significant bit (0x80) of byte 0 (the first byte) is set to 1 for long header packets.
Augmented Packet Header Diagrams: QUIC example

Uses structured, but idiomatic, text to provide constraints and model parsing context use

| T | R | P |  
|----------------------------------------|
| Version  
| DCID Len |  
| Destination Connection ID (DCID)  
| SCID Len |  
| Source Connection ID (SCID)  
| Token Length |  
| Payload  

where:

Header Form (HF): 1 bit; HF == 1. The most significant bit (0x80) of byte 0 (the first byte) is set to 1 for long header packets.

...  

DCID Len (DLen): 1 byte; DLen <= 20. This field contains the length, in bytes, of the Destination Connection ID field that follows it.

Destination Connection ID (DCID): DLen bytes. The Destination Connection ID field is between 0 and 20 bytes in length. On receipt, the value of DCID is stored as Initial DCID.

SCID Len (SLen): ...
Augmented Packet Header Diagrams: QUIC example

A Protected Packet is either a Protected Long Header Packet or a Protected Short Header Packet.

An Unprotected Packet is either a Long Header Packet or a Short Header Packet.

An Unprotected Packet is parsed from a Protected Packet using the remove_protection function. The remove_protection function is defined as:

```go
func remove_protection(from: Protected Packet) -> Unprotected Packet: ...
```

An Unprotected Packet is serialised to a Protected Packet using the apply_protection function. The apply_protection function is defined as:

```go
func apply_protection(to: Unprotected Packet) -> Protected Packet: ...
```
Augmented Packet Header Diagrams: QUIC example

A Protected Packet is either a Protected Long Header Packet or a Protected Short Header Packet.

Provides support for functions and context use

An Unprotected Packet is parsed from a Protected Packet using the remove_protection function. The remove_protection function is defined as:

```func remove_protection(from: Protected Packet) -> Unprotected Packet:
    ...
```

An Unprotected Packet is serialised to a Protected Packet using the apply_protection function. The apply_protection function is defined as:

```func apply_protection(to: Unprotected Packet) -> Protected Packet:
    ...
```
Augmented Packet Header Diagrams

- The format of packet header diagrams can be regularised with minimal change
- The format remains extremely close to that in common use, easing adoption
- It balances structure and uniformity, needed for machine parsing, with the flexibility needed for practical use
- Prototype tooling that supports this input format, generating the Network Packet Representation from it
Network Working Group                                    D. Crocker, Ed.
Request for Comments: 5234                   Brandenburg InternetWorking
STD: 68                                                       P. Overell
Obsoletes: 4234
THUS plc.
Category: Standards Track                                   January 2008
Augmented BNF for Syntax Specifications: ABNF

Status of This Memo
This document specifies an Internet standards track protocol for the Internet community, and requests discussion and suggestions for improvements. Please refer to the current edition of the "Internet Official Protocol Standards" (STD 1) for the standardization state and status of this protocol. Distribution of this memo is unlimited.

Abstract
Internet technical specifications often need to define a formal syntax. Over the years, a modified version of Backus-Naur Form (BNF), called Augmented BNF (ABNF), has been popular among many Internet specifications. The current specification documents ABNF. It balances compactness and simplicity with reasonable representational power. The differences between standard BNF and ABNF involve naming rules, repetition, alternatives, order-independence, and value ranges. This specification also supplies additional rule definitions and encoding for a core lexical analyzer of the type common to several Internet specifications.

Network Working Group                                    S. McQuistin
Internet-Draft                                                   V. Band
Intended status: Experimental                                   D. Jacob
Expires: 19 December 2020                                  C. S. Perkins
University of Glasgow
17 June 2020
Describing QUIC’s Protocol Data Units with Augmented Packet Header Diagrams
draft-mcquistin-quic-augmented-diagrams-01

Abstract
This document describes the core transport protocol data units used in the QUIC protocol using a machine-readable augmented packet header diagram format. It is intended as an example of the packet header diagram language, and not as a description of the implementation of the QUIC protocol.

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 19 December 2020.

Copyright Notice
Copyright (c) 2020 IETF Trust and the persons identified as the document authors. All rights reserved.

Automatic parser generation provides a number of opportunities to improve security
Parser Generators

- The Network Packet Representation can be used to generate implementation code in any number of target programming languages

- Core code generation functions can be implemented once, easing the development of code generators for new languages
QUIC example

An Initial Packet is formatted as follows:

<p>| |</p>
<table>
<thead>
<tr>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0</td>
</tr>
<tr>
<td>----------------------------</td>
</tr>
<tr>
<td>T</td>
</tr>
<tr>
<td>----------------------------</td>
</tr>
<tr>
<td>DCID Len</td>
</tr>
<tr>
<td>----------------------------</td>
</tr>
<tr>
<td>SCID Len</td>
</tr>
<tr>
<td>----------------------------</td>
</tr>
<tr>
<td>Token Length</td>
</tr>
<tr>
<td>----------------------------</td>
</tr>
<tr>
<td>Token</td>
</tr>
<tr>
<td>----------------------------</td>
</tr>
<tr>
<td>Packet Number</td>
</tr>
</tbody>
</table>

where:

- Header Flags (HF): 1 bit; HF == 1 – The most significant bit (0x80) of the packet header
An Initial Packet is formatted as follows:

<table>
<thead>
<tr>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>4</td>
<td>5</td>
<td>6</td>
<td>7</td>
</tr>
<tr>
<td>8</td>
<td>9</td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>

---

- Emit types and parser combinator functions for each field type
QUIC example

Emit types and parser combinator functions for structures

An Initial Packet

<table>
<thead>
<tr>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>3</td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>4</td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>5</td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>6</td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>7</td>
<td></td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>8</td>
<td></td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>9</td>
<td></td>
<td></td>
</tr>
<tr>
<td>9</td>
<td>0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>1</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- **Version**
- **DCID Len**
- **Destination Connection ID (DCID)**
- **SCID Len**
- **Source Connection ID (SCID)**
- **Token Length**
- **Token**
- **Length**
- **Packet Number**
- **Payload**

where: 46

Header Form (HE): 1 bit; HE = 1. The most significant bit (0x80) of the length field (Length) is set.
A Protected Packet is either a Protected Long Header Packet or a Protected Short Header Packet.

An Unprotected Packet is either a Long Header Packet or a Short Header Packet.

An Unprotected Packet is parsed from a Protected Packet using the remove_protection function. The remove_protection function is defined as:

```
func remove_protection(from: Protected Packet) -> Unprotected Packet:
    ...
```

An Unprotected Packet is serialised to a Protected Packet using the apply_protection function. The apply_protection function is defined as:

Generate stubs for functions
Parser Generators

- Support for different parser models — like parser combinators — can be implemented once

- This has implications for security: modern systems languages, like Rust, can be easily supported, encouraging their adoption and use

- Our prototype tooling supports Rust code generation
Augmented BNF for Syntax Specifications: ABNF

Status of This Memo

This document specifies an Internet standards track protocol for the Internet community, and requests discussion and suggestions for improvements. Please refer to the current edition of the "Internet Official Protocol Standards" (STD 1) for the standardization state and status of this protocol. Distribution of this memo is unlimited.

Abstract

Internet technical specifications often need to define a formal syntax. Over the years, a modified version of Backus-Naur Form (BNF), called Augmented BNF (ABNF), has been popular among many Internet specifications. The current specification documents ABNF. It balances compactness and simplicity with reasonable representational power. The differences between standard BNF and ABNF involve naming rules, repetition, alternatives, order-independence, and value ranges. This specification also supplies additional rule definitions and encoding for a core lexical analyzer of the type common to several Internet specifications.

Network Packet Representation

Programming Languages

- Python
- Java
- C

Network Working Group

D. Crocker, Ed.
Brandenburg InternetWorking

Request for Comments: 5234

Obsoletes: 4234

THUS plc.

Category: Standards Track

January 2008

Augmented BNF for Syntax Specifications: ABNF

Status of This Memo

This document specifies an Internet standards track protocol for the Internet community, and requests discussion and suggestions for improvements. Please refer to the current edition of the "Internet Official Protocol Standards" (STD 1) for the standardization state and status of this protocol. Distribution of this memo is unlimited.

Abstract

Internet technical specifications often need to define a formal syntax. Over the years, a modified version of Backus-Naur Form (BNF), called Augmented BNF (ABNF), has been popular among many Internet specifications. The current specification documents ABNF. It balances compactness and simplicity with reasonable representational power. The differences between standard BNF and ABNF involve naming rules, repetition, alternatives, order-independence, and value ranges. This specification also supplies additional rule definitions and encoding for a core lexical analyzer of the type common to several Internet specifications.
Example: Describing TCP

Prototype and example drafts:
http://dx.doi.org/10.5525/gla.researchdata.1083
TCP example

<p>| | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Source Port</td>
<td>Destination Port</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Sequence Number</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Acknowledgment Number</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Data</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Offset</td>
<td>Rsvd</td>
<td>W</td>
<td>C</td>
<td>R</td>
</tr>
<tr>
<td></td>
<td></td>
<td>R</td>
<td>E</td>
<td>G</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Checksum</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Urgent Pointer</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>[Options]</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

where:

Source Port: 16 bits. The source port number.

Destination Port: 16 bits. The destination port number.

Sequence Number: 32 bits. The sequence number of the first data octet in this segment (except when the SYN flag is set). If SYN is set the sequence number is the initial sequence number (ISN) and the first data octet is ISN+1.
<table>
<thead>
<tr>
<th>Source Port</th>
<th>Destination Port</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sequence Number</td>
<td></td>
</tr>
<tr>
<td>Acknowledgment Number</td>
<td></td>
</tr>
<tr>
<td>Data</td>
<td></td>
</tr>
<tr>
<td>Offset</td>
<td>Rsrvd</td>
</tr>
<tr>
<td>Window Size</td>
<td></td>
</tr>
<tr>
<td>Checksum</td>
<td>Urgent Pointer</td>
</tr>
<tr>
<td>[Options]</td>
<td></td>
</tr>
<tr>
<td>Payload</td>
<td></td>
</tr>
</tbody>
</table>

where:

**Source Port**: 16 bits. The source port number.

**Destination Port**: 16 bits. The destination port number.

**Sequence Number**: 32 bits. The sequence number of the first data octet in this segment (except when the SYN flag is set). If SYN is set the sequence number is the initial sequence number (ISN) and the first data octet is ISN+1.

**Acknowledgment Number**: 32 bits. If the ACK control bit is set, this field contains the value of the next sequence number the sender of the segment is expecting to receive. Once a connection is established, this is always sent.

**Data Offset (Offset)**: 4 bits. The number of 32 bit words in the TCP Header. This indicates where the data begins. The TCP header

---

---

**Figure 1: TCP Header Format**

Each of the TCP header fields is described as follows:

**Source Port**: 16 bits

The source port number.

**Destination Port**: 16 bits

The destination port number.

**Sequence Number**: 32 bits

The sequence number of the first data octet in this segment (except when the SYN flag is set). If SYN is set the sequence number is the initial sequence number (ISN) and the first data octet is ISN+1.
Our format retains the overall structure and layout

where:

Source Port: 16 bits. The source port number.

Destination Port: 16 bits. The destination port number.

Sequence Number: 32 bits. The sequence number of the first data octet in this segment (except when the SYN flag is set). If SYN is set, the sequence number is the initial sequence number (ISN) and the first data octet is ISN+1.

Acknowledgment Number: 32 bits. If the ACK control bit is set, this field contains the value of the next sequence number the sender of the segment is expecting to receive. Once a connection is established, this is always sent.

Data Offset (Offset): 4 bits. The number of 32-bit words in the TCP Header. While indicative where the data begins, this is not the actual length of the TCP header.
Options: [TCP Option]; Options#Size == (DOffset-5)*32; present only when DOffset > 5. Options may occupy space at the end of the TCP header and are a multiple of 8 bits in length. All options are included in the checksum.

Options: variable

Options may occupy space at the end of the TCP header and are a multiple of 8 bits in length. All options are included in the checksum. An option may begin on any octet boundary. There are two cases for the format of an option:

Case 1: A single octet of option-kind.

Case 2: An octet of option-kind (Kind), an octet of option-length, and the actual option-data octets.

The option-length counts the two octets of option-kind and option-length as well as the option-data octets.

Note that the list of options may be shorter than the data offset field might imply. The content of the header beyond the End-of-Option option must be header padding (i.e., zero).
The format requires precision, which can improve documents.
3. TCP Options

A TCP Option is one of: a EOL Option, a NOOP Option, a Maximum Segment Size Option, a Window Scale Factor Option, a Timestamp Option, or a SACK Permitted Option.

An EOL Option is formatted as follows:

0 1 2 3
0 1 2 3
+----------+
| 0 |
+----------+

where:

Option Kind (Kind): 1 byte; Kind == 0. This option code indicates the end of the option list.

A NOOP Option is formatted as follows:

0 1 2 3
0 1 2 3
+----------+
| 1 |
+----------+

where:

Option Kind (Kind): 1 byte; Kind == 1. This option code can be used between options, for example, to align the beginning of a subsequent option on a word boundary.

A Maximum Segment Size Option is formatted as follows:

0 1 2 3
0 1 2 3
+----------+
| 2 | max seg size |
+----------+

where:

Option Kind (Kind): 1 byte; Kind == 2. Length=4

Specific Option Definitions

End of Option List

+----------+
| 00000000 |
+----------+

Kind=0

This option code indicates the end of the option list. This might not coincide with the end of the TCP header according to the Data Offset field. This is used at the end of all options, not the end of each option, and need only be used if the end of the options would not otherwise coincide with the end of the TCP header.

No-Operation

+----------+
| 00000001 |
+----------+

Kind=1

This option code can be used between options, for example, to align the beginning of a subsequent option on a word boundary. There is no guarantee that senders will use this option, so receivers MUST be prepared to process options even if they do not begin on a word boundary (MUST-64).

Maximum Segment Size (MSS)

+----------+
| 00000010 00000100 | max seg size |
+----------+

Kind=2 Length=4

Maximum Segment Size Option Data: 16 bits
3. TCP Options

A TCP Option is one of: a EOL Option, a NOOP Option, a Maximum Segment Size Option, a Window Scale Factor Option, a Timestamp Option, or a SACK Permitted Option.

An EOL Option is formatted as follows:

```
0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-
| Kind=0 |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-
```

where:

Option Kind (Kind): 1 byte; Kind == 0. This option code indicates the end of the option list.

A NOOP Option is formatted as follows:

```
0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-
| Kind=1 |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-
```

where:

Option Kind (Kind): 1 byte; Kind must be 1, this option code indicates the beginning of a subsequent option on a word boundary. There is no guarantee that receivers will use this option, so receivers MUST be prepared to process options even if they do not begin on a word boundary (MUST-64).

A Maximum Segment Size Option is formatted as follows:

```
0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-
| Kind=2 |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-
```

where:

Option Kind (Kind): 1 byte; Kind must be 2, this option code indicates the beginning of a subsequent option on a word boundary.

Specific Option Definitions

End of Option List

```
+-------+
|00000000|
+-------+
Kind=0
```

This option code indicates the end of the option list. This might not coincide with the end of the TCP header according to the Data Offset field. This is used at the end of all options, not the end of each option, and need only be used if the end of the options would not otherwise coincide with the end of the TCP header.

No-Operation

```
+-------+
|00000001|
+-------+
Kind=1
```

This option code can be used between options, for example, to align the beginning of a subsequent option on a word boundary. There is no guarantee that receivers will use this option, so receivers MUST be prepared to process options even if they do not begin on a word boundary (MUST-64).

Maximum Segment Size (MSS)

```
+----------------------------------+
|00000000000000000000000000000000|
+----------------------------------+
Kind=3
```

This option code indicates the maximum segment size of a connection. This option code may be used to indicate an initial maximum segment size for a connection, or to modify the advertised maximum segment size of a connection. The maximum segment size must be advertised in all active segments. There is no guarantee that receivers will use this option, so receivers MUST be prepared to process options even if they do not begin on a word boundary (MUST-64).
Document → Network Packet Representation

<table>
<thead>
<tr>
<th>0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>Source Port</td>
</tr>
<tr>
<td>Destination Port</td>
</tr>
<tr>
<td>Sequence Number</td>
</tr>
<tr>
<td>Acknowledgment Number</td>
</tr>
<tr>
<td>Data</td>
</tr>
<tr>
<td>CEUAPRSF</td>
</tr>
<tr>
<td>Offset</td>
</tr>
<tr>
<td>Rsrvd WCSRCSYIN</td>
</tr>
<tr>
<td>Window Size</td>
</tr>
<tr>
<td>REGKHTN</td>
</tr>
<tr>
<td>Checksum</td>
</tr>
<tr>
<td>Urgent Pointer</td>
</tr>
<tr>
<td>[Options]</td>
</tr>
<tr>
<td>:</td>
</tr>
<tr>
<td>:</td>
</tr>
<tr>
<td>:</td>
</tr>
<tr>
<td>:</td>
</tr>
<tr>
<td>Payload</td>
</tr>
<tr>
<td>:</td>
</tr>
<tr>
<td>:</td>
</tr>
</tbody>
</table>

where:

Source Port: 16 bits. The source port number.

Destination Port: 16 bits. The destination port number.

Sequence Number: 32 bits. The sequence number of the first data octet in this segment (except when the SYN flag is set). If SYN is set the sequence number is the initial sequence number (ISN) and the first data octet is ISN+1.
Document → Network Packet Representation

Bit String: SourcePort

Source Port: 16 bits. The source port number.

Destination Port: 16 bits. The destination port number.

Sequence Number: 32 bits. The sequence number of the first data octet in this segment (except when the SYN flag is set). If SYN is set the sequence number is the initial sequence number (ISN) and the first data octet is ISN+1.
Document → Network Packet Representation

Bit String: SourcePort

Bit String: DestinationPort

where:

Source Port: 16 bits. The source port number.

Destination Port: 16 bits. The destination port number.

Sequence Number: 32 bits. The sequence number of the first data octet in this segment (except when the SYN flag is set). If SYN is set the sequence number is the initial sequence number (ISN) and the first data octet is ISN+1.
### Network Packet Representation

<table>
<thead>
<tr>
<th>Bit String: SourcePort</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bit String: DestinationPort</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>++++++++ bit string: Source Port</td>
<td>++++++++ bit string: Destination Port</td>
<td></td>
<td></td>
</tr>
<tr>
<td>++++++++ bit string: Sequence Number</td>
<td>++++++++ bit string: Acknowledgment Number</td>
<td></td>
<td></td>
</tr>
<tr>
<td>++++++++ bit string: Data</td>
<td>++++++++ bit string: Reserved</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Offset</td>
<td>Reserved</td>
<td>Window Size</td>
<td></td>
</tr>
<tr>
<td>C'E</td>
<td>U</td>
<td>A</td>
<td>P</td>
</tr>
<tr>
<td>++++++++ bit string: Checksum</td>
<td>++++++++ bit string: Urgent Pointer</td>
<td></td>
<td></td>
</tr>
<tr>
<td>++++++++ bit string: Options</td>
<td>++++++++ bit string: Payload</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

where:

- **Source Port**: 16 bits. The source port number.
- **Destination Port**: 16 bits. The destination port number.
- **Sequence Number**: 32 bits. The sequence number of the first data octet in this segment (except when the SYN flag is set). If SYN is set the sequence number is the initial sequence number (ISN) and the first data octet is ISN+1.
Bit String: SourcePort

Bit String: DestinationPort

where:

Source Port: 16 bits. The source port number.

Destination Port: 16 bits. The destination port number.

Sequence Number: 32 bits. The sequence number of the first data octet in this segment (except when the SYN flag is set). If SYN is set the sequence number is the initial sequence number (ISN) and the first data octet is ISN+1.
### Network Packet Representation

<table>
<thead>
<tr>
<th>Offset</th>
<th>Bit String</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Data</td>
</tr>
<tr>
<td>1</td>
<td>Rsvd</td>
</tr>
<tr>
<td>2</td>
<td>ECUAPRSF</td>
</tr>
<tr>
<td>3</td>
<td>WCRCSYI</td>
</tr>
<tr>
<td></td>
<td>REGKHTN</td>
</tr>
<tr>
<td></td>
<td>Window Size</td>
</tr>
<tr>
<td></td>
<td>Checksum</td>
</tr>
<tr>
<td></td>
<td>Urgent Pointer</td>
</tr>
<tr>
<td></td>
<td>Options</td>
</tr>
<tr>
<td></td>
<td>Payload</td>
</tr>
<tr>
<td></td>
<td>Payload</td>
</tr>
<tr>
<td></td>
<td>Payload</td>
</tr>
<tr>
<td></td>
<td>Payload</td>
</tr>
<tr>
<td></td>
<td>Payload</td>
</tr>
</tbody>
</table>

Where:

- **Source Port**: 16 bits. The source port number.
- **Destination Port**: 16 bits. The destination port number.
- **Sequence Number**: 32 bits. The sequence number of the first data octet in this segment (except when the SYN flag is set). If SYN is set the sequence number is the initial sequence number (ISN) and the first data octet is ISN+1.
Document → Network Packet Representation

| Bit String: SourcePort |
| Bit String: DestinationPort |

where:

Source Port: 16 bits. The source port number.

Destination Port: 16 bits. The destination port number.

Sequence Number: 32 bits. The sequence number of the first data octet in this segment (except when the SYN flag is set). If SYN is set, the sequence number is the initial sequence number (ISN) and the first data octet is ISN+1.
Document → Network Packet Representation

| Bit String: SourcePort |
| Bit String: DestinationPort |

<table>
<thead>
<tr>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>+++++++++++++++++++++++++++++++++++++++++++++++++++</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Source Port</td>
<td>Destination Port</td>
<td></td>
<td></td>
</tr>
<tr>
<td>+++++++++++++++++++++++++++++++++++++++++++++++++++</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Sequence Number</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Acknowledgment Number</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>+++++++++++++++++++++++++++++++++++++++++++++++++++</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Data</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Offset</td>
<td>Rsrvd</td>
<td>W</td>
<td>C</td>
</tr>
<tr>
<td>+++++++++++++++++++++++++++++++++++++++++++++++++++</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Checksum</td>
<td>Urgent Pointer</td>
<td></td>
<td></td>
</tr>
<tr>
<td>+++++++++++++++++++++++++++++++++++++++++++++++++++</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>[Options]</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>+++++++++++++++++++++++++++++++++++++++++++++++++++</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Payload</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>:</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>:</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>+++++++++++++++++++++++++++++++++++++++++++++++++++</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

where:

Source Port: 16 bits. The source port number.

Destination Port: 16 bits. The destination port number.

Sequence Number: 32 bits. The sequence number of the first data octet in this segment (except when the SYN flag is set). If SYN is set, the sequence number is the initial sequence number (ISN) and the first data octet is ISN+1.
### Bit String: SourcePort
- **Source Port**: 16 bits. The source port number.

### Bit String: DestinationPort
- **Destination Port**: 16 bits. The destination port number.

### Sequence Number
- **Sequence Number**: 32 bits. The sequence number of the first data octet in this segment (except when the SYN flag is set). If SYN is set the sequence number is the initial sequence number (ISN) and the first data octet is ISN+1.
Bit String: SourcePort

Bit String: DestinationPort

where:

Source Port: 16 bits. The source port number.

Destination Port: 16 bits. The destination port number.

Sequence Number: 32 bits. The sequence number of the first data octet in this segment (except when the SYN flag is set). If SYN is set the sequence number is the initial sequence number (ISN) and the first data octet is ISN+1.
Document → Network Packet Representation

Bit String: SourcePort

Bit String: DestinationPort

Data | C|E|U|A|P|R|S|F
Offset | Rsvd | W|C|R|C|S|S|Y|I | Window Size
| | R|E|G|K|H|T|N|N |

where:

Source Port: 16 bits. The source port number.

Destination Port: 16 bits. The destination port number.

Sequence Number: 32 bits. The sequence number of the first data octet in this segment (except when the SYN flag is set). If SYN is set the sequence number is the initial sequence number (ISN) and the first data octet is ISN+1.
### Network Packet Representation

**Bit String:**

- **Source Port**
- **Destination Port**
- **Sequence Number**
- **Acknowledgment Number**
- **Data**
- **Offset**
- **Rsvd**
- **Window Size**
- **Checksum**
- **Urgent Pointer**
- **Options**
- **Payload**

---

**where:**

- **Source Port:** 16 bits. The source port number.
- **Destination Port:** 16 bits. The destination port number.
- **Sequence Number:** 32 bits. The sequence number of the first data octet in this segment (except when the SYN flag is set). If SYN is set the sequence number is the initial sequence number (ISN) and the first data octet is ISN+1.
### Network Packet Representation

<table>
<thead>
<tr>
<th>0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>Source Port</td>
</tr>
<tr>
<td>Destination Port</td>
</tr>
<tr>
<td>Sequence Number</td>
</tr>
<tr>
<td>Acknowledgment Number</td>
</tr>
<tr>
<td>Data</td>
</tr>
<tr>
<td>Offset</td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td>Checksum</td>
</tr>
<tr>
<td>[Options]</td>
</tr>
<tr>
<td>Payload</td>
</tr>
</tbody>
</table>

where:

**Source Port**: 16 bits. The source port number.

**Destination Port**: 16 bits. The destination port number.

**Sequence Number**: 32 bits. The sequence number of the first data octet in this segment (except when the SYN flag is set). If SYN is set the sequence number is the initial sequence number (ISN) and the first data octet is ISN+1.
# Network Packet Representation

<p>| | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td></td>
</tr>
<tr>
<td>0 1 2</td>
<td>3 4 5</td>
<td>6 7 8</td>
<td>9 0 1</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Source Port</th>
<th>Destination Port</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Sequence Number</th>
<th>Acknowledgment Number</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Bit String: SourcePort</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 1 2 3 4 5 6 7 8 9 0 1</td>
</tr>
<tr>
<td>++++++++++++++++++++++++</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Bit String: DestinationPort</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 1 2 3 4 5 6 7 8 9 0 1</td>
</tr>
<tr>
<td>++++++++++++++++++++++++</td>
</tr>
</tbody>
</table>

**Data**

- C: Control
- E: Echo
- U: Urgent
- A: Acknowledgment
- P: Push
- R: Reserve
- S: Syn
- F: Fin

**Offset**

- Rsrvd: Reserved
- WC: Window Control
- CR: Congestion Window

**Window Size**

- EG: End of Segment
- KH: Keep Alive
- TN: Time Stamp

**Checksum**

- Options

**Payload**

- Source Port: 16 bits. The source port number.

- Destination Port: 16 bits. The destination port number.

- Sequence Number: 32 bits. The sequence number of the first data octet in this segment (except when the SYN flag is set). If SYN is set, the sequence number is the initial sequence number (ISN) and the first data octet is ISN+1.
Document → Network Packet Representation

Bit String: SourcePort

Bit String: DestinationPort

<table>
<thead>
<tr>
<th>0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>Source Port</td>
</tr>
<tr>
<td>Destination Port</td>
</tr>
<tr>
<td>Sequence Number</td>
</tr>
<tr>
<td>Acknowledgment Number</td>
</tr>
<tr>
<td>Data</td>
</tr>
<tr>
<td>Offset Rsvd</td>
</tr>
<tr>
<td>RESKHTN</td>
</tr>
<tr>
<td>Checksum</td>
</tr>
<tr>
<td>Urgent Pointer</td>
</tr>
<tr>
<td>[Options]</td>
</tr>
<tr>
<td>Payload</td>
</tr>
</tbody>
</table>

where:

Source Port: 16 bits. The source port number.

Destination Port: 16 bits. The destination port number.

Sequence Number: 32 bits. The sequence number of the first data octet in this segment (except when the SYN flag is set). If SYN is set the sequence number is the initial sequence number (ISN) and the first data octet is ISN+1.
Document → Network Packet Representation

| Bit String: SourcePort |
| Bit String: DestinationPort |

where:

Source Port: 16 bits. The source port number.

Destination Port: 16 bits. The destination port number.

Sequence Number: 32 bits. The sequence number of the first data octet in this segment (except when the SYN flag is set). If SYN is set the sequence number is the initial sequence number (ISN) and the first data octet is ISN+1.
<table>
<thead>
<tr>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
</tr>
</tbody>
</table>

Bit String: SourcePort

Bit String: DestinationPort

**Bit String:**

- **Source Port:** 16 bits. The source port number.
- **Destination Port:** 16 bits. The destination port number.
- **Sequence Number:** 32 bits. The sequence number of the first data octet in this segment (except when the SYN flag is set). If SYN is set the sequence number is the initial sequence number (ISN) and the first data octet is ISN+1.
### Network Packet Representation

<p>| | | | | | | | | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>7</td>
<td>8</td>
<td>9</td>
<td>0</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- **Source Port**: 16 bits. The source port number.

- **Destination Port**: 16 bits. The destination port number.

- **Sequence Number**: 32 bits. The sequence number of the first data octet in this segment (except when the SYN flag is set). If SYN is set the sequence number is the initial sequence number (ISN) and the first data octet is ISN+1.

### Bit String

- **Source Port**
- **Destination Port**
Document → Network Packet Representation

Bit String: SourcePort

Bit String: DestinationPort

where:

Source Port: 16 bits. The source port number.

Destination Port: 16 bits. The destination port number.

Sequence Number: 32 bits. The sequence number of the first data octet in this segment (except when the SYN flag is set). If SYN is set the sequence number is the initial sequence number (ISN) and the first data octet is ISN+1.
Document → Network Packet Representation

Bit String: SourcePort

Bit String: DestinationPort

Options: [TCP Option]:  Options#Size == (DOffset-5)*32; present only when DOffset > 5. Options may occupy space at the end of the TCP header and are a multiple of 8 bits in length. All options are included in the checksum.
Document → Network Packet Representation

<p>| | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td></td>
</tr>
</tbody>
</table>

Bit String: SourcePort

Bit String: DestinationPort

Data | [CEUAPR] | [CEUAPR] | [CEUAPR] | [CEUAPR] | [CEUAPR] |
Offset | Rsrvd | [WSCR] | [SCSY] | Window Size
| [REGH] | [HTNN] |

Checksum | Urgent Pointer

[Options]

Payload

where:

Options: [TCP Option], Options#Size = (DOffset-5)*32; present only when DOffset > 5. Options may occupy space at the end of the TCP header and are a multiple of 8 bits in length. All options are included in the checksum.
3. TCP Options

A TCP Option is one of: a EOL Option, a NOOP Option, a Maximum Segment Size Option, a Window Scale Factor Option, a Timestamp Option, or a SACK Permitted Option.

An EOL Option is formatted as follows:

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

where:

Option Kind (Kind): 1 byte; Kind == 0. This option code indicates the end of the option list.

A NOOP Option is formatted as follows:

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

where:

Option Kind (Kind): 1 byte; Kind == 1. This option code can be used between options, for example, to align the beginning of a subsequent option on a word boundary.
3. TCP Options

A TCP Option is one of: a EOL Option, a NOOP Option, a Maximum Segment Size Option, a Window Scale Factor Option, a Timestamp Option, or a SACK Permitted Option.

An EOL Option is formatted as follows:

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

where:

Option Kind (Kind): 1 byte; Kind == 0. This option code indicates the end of the option list.

A NOOP Option is formatted as follows:

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

where:

Option Kind (Kind): 1 byte; Kind == 1. This option code can be used between options, for example, to align the beginning of a
3. TCP Options

A TCP Option is one of: a EOL Option, a NOOP Option, a Maximum Segment Size Option, a Window Scale Factor Option, a Timestamp Option, or a SACK Permitted Option.

An EOL Option 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
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
  | Kind |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

where:

Option Kind (Kind): 1 byte; Kind == 0. This option code indicates the end of the option list.

A NOOP Option 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
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
  | Kind |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

where:

Option Kind (Kind): 1 byte; Kind == 1. This option code can be used between options, for example, to align the beginning of a
3. TCP Options

A TCP Option is one of: a EOL Option, a NOOP Option, a Maximum Segment Size Option, a Window Scale Factor Option, a Timestamp Option, or a SACK Permitted Option.

An EOL Option 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
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|    0                                    |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

where:

**Option Kind (Kind)**: 1 byte; Kind == 0. This option code indicates the end of the option list.

A NOOP Option 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
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|    1                                    |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

where:

**Option Kind (Kind)**: 1 byte; Kind == 1. This option code can be used between options, for example, to align the beginning of a
3. TCP Options

A TCP Option is one of: a EOL Option, a NOOP Option, a Maximum Segment Size Option, a Window Scale Factor Option, a Timestamp Option, or a SACK Permitted Option.

An EOL Option is formatted as follows:

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

where:

**Option Kind (Kind)** 1 byte; Kind == 0. This option code indicates the end of the option list.

A NOOP Option is formatted as follows:

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

where:

**Option Kind (Kind)**: 1 byte; Kind == 1. This option code can be used between options, for example, to align the beginning of a
3. TCP Options

A TCP Option is one of: a EOL Option, a NOOP Option, a Maximum Segment Size Option, a Window Scale Factor Option, a Timestamp Option, or a SACK Permitted Option.

An EOL Option is formatted as follows:

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

where:

Option Kind (Kind): 1 byte; **Kind == 0**. This option code indicates the end of the option list.

A NOOP Option is formatted as follows:

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

where:

Option Kind (Kind): 1 byte; **Kind == 1**. This option code can be used between options, for example, to align the beginning of a
3. TCP Options

A TCP Option is one of: a EOL Option, a NOOP Option, a Maximum Segment Size Option, a Window Scale Factor Option, a Timestamp Option, or a SACK Permitted Option.

An EOL Option is formatted as follows:

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

where:

Option Kind (Kind): 1 byte; Kind == 0. This option code indicates the end of the option list.

A NOOP Option is formatted as follows:

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

where:

Option Kind (Kind): 1 byte; Kind == 1. This option code can be used between options, for example, to align the beginning of a
3. TCP Options

A TCP Option is one of: a EOL Option, a NOOP Option, a Maximum Segment Size Option, a Window Scale Factor Option, a Timestamp Option, or a SACK Permitted Option.

An EOL Option is formatted as follows:

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

where:

Option Kind (Kind): 1 byte; Kind == 0. This option code indicates the end of the option list.

A NOOP Option is formatted as follows:

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

where:

Option Kind (Kind): 1 byte; Kind == 1. This option code can be used between options, for example, to align the beginning of a
A TCP Option is one of: a EOL Option, a NOOP Option, a Maximum Segment Size Option, a Window Scale Factor Option, a Timestamp Option, or a SACK Permitted Option.
A TCP Option is one of: a EOL Option, a NOOP Option, a Maximum Segment Size Option, a Window Scale Factor Option, a Timestamp Option, or a SACK Permitted Option.
**Network Packet Representation**

Bit String: SourcePort

Bit String: DestinationPort

... 

Enum: TCPOption

Options: [TCP Option]: Options#Size == (DOffset-5)*32; present only when DOffset > 5. Options may occupy space at the end of the TCP header and are a multiple of 8 bits in length. All options are included in the checksum.
Document → Network Packet Representation

Bit String: SourcePort

Bit String: DestinationPort

Array: Options

Options: [TCP Option]. Options#Size == (DOffset-5)*32; present only when DOffset > 5. Options may occupy space at the end of the TCP header and are a multiple of 8 bits in length. All options are included in the checksum.
Document → Network Packet Representation

<table>
<thead>
<tr>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>0</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>7</td>
<td>8</td>
<td>9</td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>
| +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
| | Source Port | Destination Port |
| +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
| | Sequence Number |
| +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
| | Acknowledgment Number |
| +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
| | Data | | C | E | U | A | P | R | S | F |
| Offset | Rsvd | W | C | R | C | S | S | Y | I |
| | R | E | G | K | H | T | N | N |
| +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
| | Checksum | Urgent Pointer |
| +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
| [Options] |
| [OPOption] |
| Array: Options |

where:

Options: [TCP Option]. Options#Size == (DOffset-5)*32; present only when DOffset > 5. Options may occupy space at the end of the TCP header and are a multiple of 8 bits in length. All options are included in the checksum.
Document → Network Packet Representation

<table>
<thead>
<tr>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>++++++++---------------------------+ ++++++++---------------------------+</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Source Port</td>
<td>Destination Port</td>
<td></td>
<td></td>
</tr>
<tr>
<td>++++++++---------------------------+ ++++++++---------------------------+</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Sequence Number</td>
<td>Acknowledgment Number</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Bit String: SourcePort

Bit String: DestinationPort

\[\ldots\]

Array: Options

Bit String: Payload

where:

Options: [TCP Option]; Options#Size == (DOffset-5)*32; present only when DOffset > 5. Options may occupy space at the end of the TCP header and are a multiple of 8 bits in length. All options are included in the checksum.
Bit String: SourcePort

Bit String: DestinationPort

Struct: TCPHeader

Array: Options

Bit String: Payload

where:

Options: [TCP Option]; Options#Size == (DOffset-5)*32; present only when DOffset > 5. Options may occupy space at the end of the TCP header and are a multiple of 8 bits in length. All options are included in the checksum.
This document describes the TCP protocol. The TCP protocol uses TCP Headers.

- **Bit String: SourcePort**
- **Bit String: DestinationPort**
- **Enum: TCPOption**
- **Array: Options**
- **Bit String: Payload**
- **Struct: TCPHeader**
This document describes the TCP protocol. The TCP protocol uses TCP Headers.
Network Packet Representation $\rightarrow$ Rust code
Network Packet Representation → Rust code

```rust
// Context
pub struct Context {
    pub data_size: u32,
}
```
Network Packet Representation → Rust code

```rust
#[derive(Debug, PartialEq, Eq)]
pub struct TcpHeaderSourcePort(pub u16);

pub fn parse_tcp_header_source_port<'a>(input: &'a [u8], usize, context: &'a mut Context)
    -> IResult<&'a [u8], usize, TcpHeaderSourcePort, &'a mut Context> {
    (take(16 as usize)(input).map(|(i, o)| (i, TcpHeaderSourcePort(o))), context)
}
```
Network Packet Representation → Rust code

```rust
data struct NoOpOptionKind;

struct EOLOption {
    option_kind: EOLOptionOptionKind,
}

fn parse_eol_option<T: Parse<Error>>() (mut input: &[u8], size: usize, mut context: &mut Context) -> Result<(), EOLOption> {
    let option_kind = match parse_eol_option_option_kind(input, context) {
        Ok(i) => {
            input = i;
            context = c;
            o
        },
        Err(e) => return Err(e),
    };
    return Result::Err(Error::ParseError((input, ErrorKind::NonEmpty)));
}
```
Network Packet Representation → Rust code

EOLOptionKind → EOLOption
NoOpOptionKind → NoOpOption
TCPOption
SourcePort
DestinationPort
Context
TCPHeader
Options
Payload

```rust
#[derive(Debug)]
pub enum TcpOption {
    EolOption(EOLOption),
    NoOpOption(NoOpOption),
}

pub fn parse_tcp_option<'a>(input: &'a [u8], usize, mut context: &'a mut Context) -> Result<(), ErrorKind> {
    match parse_eol_option(input, context) {
        (Ok((i, o)), c) => {
            context = c;
            (Ok((i, TcpOption::EolOption(o))), c)
        }
    }
    match parse_noop_option(input, context) {
        (Ok((i, o)), c) => {
            context = c;
            (Ok((i, TcpOption::NoOpOption(o))), c)
        }
    }
}
```
Network Packet Representation → Rust code
#[derive(Debug)]
pub struct TcpHeaderOptions(pub Vec<TcpOption>);

pub fn parse_tcp_header_options<'a>(mut input: (&'a [u8], usize), mut context: &'a mut Context,
data_offset: usize)
  → (IResult<&'a [u8], usize>, TcpHeaderOptions, &'a mut Context) {
let mut tcp_header_options = TcpHeaderOptions(Vec::new());
let mut bits_read = 0;
let bits_to_read = (data_offset - 5) * 32;
while bits_to_read > bits_read {
  match parse_tcp_option(input, context) {
    (Ok((i, o)), c) ⇒ {
      bits_read += (input.1 - i.1) + (input.0.len() - i.0.len()) * 8;
      input = i;
      context = c;
      tcp_header_options.0.push(o);
    },
    (Err(_e), c) ⇒ {
      context = c;
      break
    }
  }
}
if bits_read ≠ bits_to_read {
  (Err::Error((input, ErrorKind::NonEmpty)), context)
} else {
  (Ok((input, tcp_header_options)), context)
}
```rust
#[derive(Debug)]
pub struct TcpHeader {
    pub source_port: TcpHeaderSourcePort,
    pub destination_port: TcpHeaderDestinationPort,
    pub sequence_number: TcpHeaderSequenceNumber,
    pub acknowledgment_number: TcpHeaderAcknowledgmentNumber,
    pub data_offset: TcpHeaderDataOffset,
    pub reserved: TcpHeaderReserved,
    pub cwr: TcpHeaderCwr,
    pub ece: TcpHeaderEce,
    pub urg: TcpHeaderUrg,
    pub ack: TcpHeaderAck,
    pub psh: TcpHeaderPsh,
    pub rst: TcpHeaderRst,
    pub syn: TcpHeaderSyn,
    pub fin: TcpHeaderFin,
    pub window_size: TcpHeaderWindowSize,
    pub checksum: TcpHeaderChecksum,
    pub urgent_pointer: TcpHeaderUrgentPointer,
    pub options: Option<TcpHeaderOptions>,
    pub payload: TcpHeaderPayload,
}

pub fn parse_tcp_header<'a>(mut input: &'a [u8], mut context: &'a mut Context) {
    let source_port = match parse_tcp_header_source_port(input, context) {
        (IResult::Ok((i, o)), c) => {
            input = i;
            context = c;
            o
        }
        (IResult::Err(e), c) => return (IResult::Err(e), c),
    }
}
```
Network Packet Representation → Rust code

```rust
#[derive(Debug)]
pub enum PDU {
    TcpHeader(TcpHeader),
}

pub fn parse_pdu<'a>(input: &'a [u8], usize, mut context: &'a mut Context) -> IResult<&'a [u8], usize, PDU, &'a mut Context) {
    match parse_tcp_header(input, context) {
        (IResult::Ok(([], 0), o), c) => return (IResult::Ok(([], 0), PDU::TcpHeader(o)), c),
        (IResult::Ok(_, c)) | (IResult::Err(_, c)) => { context = c; }
    }
    (IResult::Err(Err::Error((input, ErrorKind::NonEmpty))), context)
}
```
TCP Example: Summary

- Augmented packet header diagram version of the TCP draft is largely the same as the existing format

- Representing the protocol using the Network Packet Representation provides type-checking and allows for other checks

- The Network Packet Representation easily supports the parser combinator approach, with depth-first traversal of the type tree
Conclusions

- Support for complex protocols with contextual, multi-stage parsing processes
- An incremental path to adoption within the standards community
- An important step towards the routine use of parser generating tooling, that should lead to standards that are safer and more trustworthy

ANRW ’20 paper: http://smcquistin.uk/p/mcquistin2020parsing
Code and example drafts: http://dx.doi.org/10.5525/gla.researchdata.1083