Parsing Protocol Standards to Parse Standard Protocols

Stephen McQuistin
Vivian Band
Dejice Jacob
Colin Perkins

Applied Networking Research Workshop 2020

This work is funded by the UK Engineering and Physical Sciences Research Council, under grant EP/R04144X/1.
Parsing Protocol Standards...

- 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.
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.
A common protocol representation
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
The Network Packet Representation

An RTP Data Packet is formatted as follows:

```
0 1 2 3 0 1 2 3 0 1 2 3 0 1 2 3
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|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 |OPTIONAL |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|format defined by signalling |(if X=1) |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|Payload|
|variable format and length, depends on 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:

<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>

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...

Bit strings to represent raw protocol data
With these requirements in mind, Section 4 describes the within and across different PDUs, we model a strongly typed parsing context. Secondly, not well-documented, we show the clear representation of packet headers, and we see widespread use. Such diagrams are well suited to providing "rough consensus" on the result in face-to-face and electronic forums, with the goal of reaching consensus. These ad-hoc languages lack the rigorous formal definition and being used to generate output implementations. The Network Packet Representation, a type system for representing protocols, was discussed, and drafts of standards documents are debated, in face-to-face and electronic forums, with the goal of reaching consensus. These languages have been developed, including ABNF, YANG, and variants of XML. These languages are typically not widely adopted. We believe that a common representation provides a number of other benefits, including improvements to the safety and security of different protocol implementations; Section 6 outlines these. Finally, the type system is discussed, and how different protocol description formats and techniques have been used to allow them to be parsed efficiently structured and formal to allow them to be parsed by machine, yet without a steep learning curve. This in-
With these requirements in mind, Section 4 describes the within a wider architecture, being generated by an input not well-de...
The Network Packet Representation

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) |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

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,
An RTP Data Packet is formatted as follows:

```
0 1 2 3
0 1 2 3
+--------------------------------------------------+
| Version (V): 2 bits; V == 2. This field identifies the version of RTP. |
+--------------------------------------------------+
| Padding (P): 1 bit. If the padding bit is set, the padding bit is set. |
| Sequence Number (S): 12 bits. This field identifies the sequence number. |
| Timestamp (T): 32 bits. This field identifies the timestamp. |
+--------------------------------------------------+
| Synchronization Source (SSRC) identifier (4 * CC octets) |
| Header Extension Length (4 * CC octets) |
| Header Extension Format Defined by Signalling |
+--------------------------------------------------+
| Payload |
+--------------------------------------------------+
```

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

<table>
<thead>
<tr>
<th>JRNL: The Network Packet Representation</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>An RTP Data Packet is formatted as follows:</strong></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>
</tr>
<tr>
<td>+-------------+-------------+-------------+</td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td>+-------------+-------------+-------------+</td>
</tr>
<tr>
<td>+-------------+-------------+-------------+</td>
</tr>
<tr>
<td>timestamp</td>
</tr>
<tr>
<td>+-------------+-------------+-------------+</td>
</tr>
<tr>
<td>synchronization source (SSRC) identifier</td>
</tr>
<tr>
<td>+-------------+-------------+-------------+</td>
</tr>
<tr>
<td>[CSRC identifier list]</td>
</tr>
<tr>
<td>+-------------+-------------+-------------+</td>
</tr>
<tr>
<td>(4 * CC octets)</td>
</tr>
<tr>
<td>+-------------+-------------+-------------+</td>
</tr>
<tr>
<td>CC may be zero</td>
</tr>
<tr>
<td>+-------------+-------------+-------------+</td>
</tr>
<tr>
<td>header extension</td>
</tr>
<tr>
<td>+-------------+-------------+-------------+</td>
</tr>
<tr>
<td>format defined by signalling</td>
</tr>
<tr>
<td>+-------------+-------------+-------------+</td>
</tr>
<tr>
<td>Pad (PadCnt octets, if P=1) Pad (PadCnt (if P=1))</td>
</tr>
<tr>
<td>+-------------+-------------+-------------+</td>
</tr>
<tr>
<td>where:</td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td>Version (V): 2 bits; V == 2. This field identifies the version of RTP.</td>
</tr>
<tr>
<td>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 &quot;vat&quot; audio tool.)</td>
</tr>
<tr>
<td>Padding (P): 1 bit. If the padding bit is set</td>
</tr>
</tbody>
</table>

Contextual data shared out-of-band or between different PDUs
A protocol is comprised of multiple PDUs.
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.
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
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
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

More details about the type system in the paper
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.
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:

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

An Initial Packet is formatted as follows:

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

[Diagram]

Maintains an easy-to-read diagram showing the layout of packets

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

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
def 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
def 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.

An Initial Packet is formatted as follows:

```
0 1 2 3 4 5 6 7 8 9 a b c d e f
```

<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>

```
[1][8] 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.
- 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 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.
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
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 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>
</tbody>
</table>

+++++++

<table>
<thead>
<tr>
<th>T</th>
<th>R</th>
<th>P</th>
</tr>
</thead>
</table>

+++++++

<table>
<thead>
<tr>
<th>Version</th>
</tr>
</thead>
</table>

+++++++

<table>
<thead>
<tr>
<th>DCID Len</th>
</tr>
</thead>
</table>

+++++++

<table>
<thead>
<tr>
<th>Destination Connection ID (DCID)</th>
</tr>
</thead>
</table>

+++++++

<table>
<thead>
<tr>
<th>SCID Len</th>
</tr>
</thead>
</table>

+++++++

<table>
<thead>
<tr>
<th>Source Connection ID (SCID)</th>
</tr>
</thead>
</table>

+++++++

<table>
<thead>
<tr>
<th>Token Length</th>
</tr>
</thead>
</table>

+++++++

<table>
<thead>
<tr>
<th>Token</th>
</tr>
</thead>
</table>

+++++++

<table>
<thead>
<tr>
<th>Length</th>
</tr>
</thead>
</table>

+++++++

<table>
<thead>
<tr>
<th>Packet Number</th>
</tr>
</thead>
</table>

+++++++

<table>
<thead>
<tr>
<th>Payload</th>
</tr>
</thead>
</table>

+++++++

where:

Header Flags (HF): 1 bit; HF == 1 - The most significant bit (0x80) of the Flags field is set, indicating the presence of the header flags.
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
```

<table>
<thead>
<tr>
<th>T</th>
<th>R</th>
<th>P</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- Version
- DCID Len
- Destination Connection ID (DCID)
- 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
QUIC example

Emit types and parser combinator functions for structures

An Initial Packet

where:

Header Form (HF): 1 bit; HF == 1 — The most significant bit (0x80) of the packet payload

Version

DCID Len

Destination Connection ID (DCID)

SCID Len

Source Connection ID (SCID)

Token Length

Token

Length

Packet Number

Payload

0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1

47
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:

---

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
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.
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

Paper: https://irtf.org/anrw/2020-program.html#p21