Parsing Protocol Standards

Stephen McQuistin
Colin Perkins

Multi-Service Networks workshop
5th July 2019
IETF protocol standards

- Developed by large groups of people, often remotely
- Process is iterative and incremental
- Output is a document that is mostly English prose
- No good way to automatically verify or validate a standards document
- Inconsistencies & ambiguities in specs → buggy implementations
IETF protocol standards

- Developed by large groups of people, often remotely
- Process is iterative and incremental
- Output is a document mostly in English prose
- No good way to automatically verify or validate a standards document
- Inconsistencies & ambiguities in spec → buggy implementations

.. but the process works: we have the Internet!
Improving protocol standards

- Goal: shift towards a test-driven development style approach, where running a suite of validation and verification tools over a standards document becomes commonplace

- Don’t want to replace the process, but to augment it
Describing protocol parsing

• First aim: build a tool that allows for a parser for the specified protocol to be generated automatically

• Need a machine-readable description of the protocol’s data units, and all the metadata needed to parse them

• Good place to start: knowing what the protocol looks like forms the basis of more complex tools
Design principles

• Most readers are human
• Authorship tools are diverse
• Canonical specifications
• Expressiveness
• Minimise required change
ASCII packet diagrams

TCP Header Format

<table>
<thead>
<tr>
<th>0</th>
<th>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></td>
<td>Source Port</td>
</tr>
<tr>
<td></td>
<td>Destination Port</td>
</tr>
<tr>
<td></td>
<td>Sequence Number</td>
</tr>
<tr>
<td></td>
<td>Acknowledgment Number</td>
</tr>
<tr>
<td></td>
<td>Data</td>
</tr>
<tr>
<td></td>
<td>Reserved</td>
</tr>
<tr>
<td></td>
<td>Offset</td>
</tr>
<tr>
<td></td>
<td>Reserved</td>
</tr>
<tr>
<td></td>
<td>Window</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>Padding</td>
</tr>
<tr>
<td></td>
<td>data</td>
</tr>
</tbody>
</table>

TCP Header Format

Note that one tick mark represents one bit position.

Figure 3.

Source Port: 16 bits

The source port number.

Destination Port: 16 bits

The destination port number.
ASCII packet diagrams

ASCII diagrams already specify much of the protocol’s syntax

TCP Header Format

Note that one tick mark represents one bit position.

Figure 3.

Source Port: 16 bits
The source port number.

Destination Port: 16 bits
The destination port number.
4.2. Relay Source Port Option for DHCPv6

The "Relay Source Port Option" is a new DHCPv6 option. It MUST be used by either 1) a DHCPv6 relay agent that uses a non-DHCP UDP port (not 547) communicating with the IPv6 server and the upstream relay agent or 2) an IPv6 relay agent that detects the use of a non-DHCP UDP port (not 547) by a downstream relay agent.

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

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

Where:

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

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

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

Figure 3.

Source Port: 16 bits

The source port number.

Destination Port: 16 bits

The destination port number.
4.2. Relay Source Port Option for DHCPv6

"Relay Source Port Option" is a new DHCPv6 option. It MUST be
implemented by either 1) a DHCPv6 relay agent that uses a non-DHCP
UDP port communicating with the IPv6 server and the upstream relay
agent, or 2) an IPv6 relay agent that detects the use of a non-DHCP
UDP port (not 547) by a downstream relay agent.

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

<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</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>Source Port</td>
</tr>
<tr>
<td>Destination Port</td>
</tr>
<tr>
<td>Length</td>
</tr>
<tr>
<td>Checksum</td>
</tr>
<tr>
<td>Data</td>
</tr>
<tr>
<td>User Datagram Header Format</td>
</tr>
</tbody>
</table>

Field Descriptions:

Source Port: 16 bits
- The source port number.

Destination Port: 16 bits
- The destination port number.

Length: 16 bits
- Indicates the length, in octets, of the user datagram including
  the header and the data. (This means the minimum value of the
  length is eight.)

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

Downstream Source Port: 16 bits
- To be set by the IPv6 relay agent to the downstream relay agent's
  UDP source port used for the DHCP UDP packet, or to zero if only
  the local relay agent uses the non-DHCP UDP port (not 547).
4.2. Relay Source Port Option for DHCPv6

"Relay Source Port Option" is a new DHCPv6 option. It MUST be

1) communicating with the IPv6 server and the upstream relay
2) an IPv6 relay agent that detects the use of a non-DHCP (not 547) by a downstream relay agent.

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

```
+----------+----------+
| Source Port | Destination Port |
| Length     | Checksum  |
| Data       |           |
+----------+----------+
```

4.1. Availability SCSI-TLV

The Generalized SCSI is defined in [RFC8258]. This document defines a new type of Generalized SCSI-TLV called the Availability SCSI-TLV. The Availability SCSI-TLV can be included one or more times, in the following format:

```
+----------+----------+
| Type     | Length   |
| Availability level |
| LSP Bandwidth at Availability level n |
+----------+----------+
```

Type: 0x000A, 16 bits
Length: 2 octets (16 bits)
Availability level: 32 bits

This field is a binary32-format floating-point number as defined by [IEEE754-2008]. The bytes are transmitted in network order; that is, the byte containing the sign bit is transmitted first. This field describes the decimal value of the availability guarantee of the Switching Capability Interface Switching Capability Descriptor object [RFC4201]. The value MUST be less than 1. The Availability level field is usually expressed as the value 0.99/0.999/0.9999/0.99999/...
4.2. Relay Source Port Option for DHCPv6

"Relay Source Port Option" is a new DHCPv6 option. It MUST be either 1) a DHCPv6 relay agent that uses a non-DHCP UDP port communicating with the IPv6 server and the upstream relay or 2) an IPv6 relay agent that detects the use of a non-DHCP (not 547) by a downstream relay agent.

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

<table>
<thead>
<tr>
<th>1</th>
<th>2</th>
<th>3</th>
</tr>
</thead>
<tbody>
<tr>
<td>9012345678901</td>
<td>2</td>
<td>3</td>
</tr>
</tbody>
</table>

User Datagram Header Format

**Fields**

Source Port is an optional field, when meaningful, it indicates the port of the sending process, and may be assumed to be the port to which reply should be addressed in the absence of any other information. If not used, a value of zero is inserted.

Destination Port has a meaning within the context of a particular internet destination address.

The following is the format of the MRT Capability Parameter:

<table>
<thead>
<tr>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
</tr>
</thead>
<tbody>
<tr>
<td>0123456789012345678901</td>
<td>2</td>
<td>3</td>
<td></td>
</tr>
</tbody>
</table>

MRT Capability TLV Format

Type: 0x000A, 16 bits

Length: 2 octets (16 bits)

Availability level: 32 bits

This field is a binary32-format floating-point number as defined by [IEEE754-2008]. The bytes are transmitted in network order; that is, the byte containing the sign bit is transmitted first. This field describes the decimal value of the availability guarantee of the Switching Capability Interface Switching Capability Descriptor object [RFC4201]. The value MUST be less than 1. The Availability level field is usually expressed as the value 0.999990.999990.99999.
4.2. Relay Source Port Option for DHCPv6

The "Relay Source Port Option" is a new DHCPv6 option. It MUST be implemented in either 1) a DHCPv6 relay agent that uses a non-DHCP UDP port to communicate with the IPv6 server and the upstream relay or 2) an IPv6 relay agent that detects the use of a non-DHCP (not 547) by a downstream relay agent.

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

```
+-------------+-------------+-------------+
| Source Port | Destination Port |
+-------------+-------------+-------------+
| Length      | Checksum    |
+-------------+-------------+
| Data        |
+-------------+-------------+
```

13

2. ICMP Extended Echo Request

The ICMP Extended Echo Request message is defined for both ICMPv4 and ICMPv6. Like any ICMP message, the ICMP Extended Echo Request message is encapsulated in an IP header. The ICMPv4 version of the Extended Echo Request message is encapsulated in an IPv4 header, while the ICMPv6 version is encapsulated in an IPv6 header.

Figure 1 depicts the ICMP Extended Echo Request message.

```
+-------------+-------------+-------------+
| |             |             |
| Type | Code | Checksum |
+-------------+-------------+-------------+
| Identifier |Sequence Number| Reserved |
+-------------+-------------+-------------+
| ICMP Extension Structure |
+-------------+-------------+
```

Figure 1: ICMP Extended Echo Request Message

IP Header fields:

- Source Address: The Source Address identifies the probing interface. It MUST be a valid IPv4 or IPv6 unicast address.
- Destination Address: The Destination Address identifies the proxy interface. It MUST be a unicast address.

ICMP fields:

- Type: Extended Echo Request. The value for ICMPv4 is 42. The value for ICMPv6 is 160.
- Code: MUST be set to 0 and MUST be ignored upon receipt.

Following is the format of the ICMPv4 Extended Echo Request message:

```
+-------------+-------------+-------------+
| |             |             |
| Type | Code | Checksum |
+-------------+-------------+-------------+
| Identification |Sequence Number| Reserved |
+-------------+-------------+-------------+
| ICMP Extension Structure |
+-------------+-------------+
```

Following is the format of the ICMPv6 Extended Echo Request message:

```
+-------------+-------------+-------------+
| |             |             |
| Type | Code | Checksum |
+-------------+-------------+-------------+
| Identifier |Sequence Number| Reserved |
+-------------+-------------+-------------+
| ICMP Extension Structure |
+-------------+-------------+
```

2
The FEC type for the P2MP PW Upstream FEC Element is encoded as follows:

- **P2MP PW Up:**
  - 8-bit representation for the P2MP PW Upstream FEC type.
- **C bit:**
  - A value of 1 or 0 indicating whether a control word is present or absent for the P2MP PW.

### Figure 2: P2MP PW Upstream FEC Element

#### 3.1. Availability SCSI-TLV

An e32-bit field is defined for both ICMPv4 and ICMPv6. It is the extended echo request. The ICMPv4 version of the message is encapsulated in an IPv4 header, while the ICMPv6 version is encapsulated in an IPv6 header.

#### 3.2. Echo Request Message

- **Echo Request Message:**
  - The type identifies the probing interface. It MUST be a valid IPv4 or IPv6 unicast address.
  - **Destination Address:** The Destination Address identifies the proxy interface. It MUST be a unicast address.

#### ICMP fields:

- **Type:** Extended Echo Request. The value for ICMPv4 is 42. The value for ICMPv6 is 160.

- **Code:** MUST be set to 0 and MUST be ignored upon receipt.
The FEC type for the P2MP PW Upstream FEC Element is encoded as follows:

```
<table>
<thead>
<tr>
<th>P2MP PW Up=0x82</th>
<th>PW Type</th>
<th>PW Info Length</th>
</tr>
</thead>
<tbody>
<tr>
<td>AGI Type</td>
<td>AGI Length</td>
<td>AGI Value</td>
</tr>
<tr>
<td>-------------</td>
<td>-------------</td>
<td>-------------</td>
</tr>
<tr>
<td>AGI Value (contd.)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>AII Type</td>
<td>AII Length</td>
<td>AII Value</td>
</tr>
<tr>
<td>-------------</td>
<td>-------------</td>
<td>-------------</td>
</tr>
<tr>
<td>AII Value (contd.)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>PMSI Tunnel Type</td>
<td>PMSI TT Length</td>
<td></td>
</tr>
<tr>
<td>Transport LSP ID</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Optional Parameters</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
```

Figure 2: P2MP PW Upstream FEC Element

* P2MP PW Up:
  8-bit representation for the P2MP PW Upstream FEC type.

* C bit:
  A value of 1 or 0 indicating whether a control word is present or absent for the P2MP PW.

---

3.2. Message Format

The CoAP message format defined in [RFC7252], as shown in Figure 3, relies on the datagram transport (UDP, or DTLS over UDP) for keeping the individual messages separate and for providing length information.

```
<table>
<thead>
<tr>
<th>Ver</th>
<th>T</th>
<th>TKL</th>
<th>Code</th>
<th>Message ID</th>
</tr>
</thead>
<tbody>
<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>
```

Figure 3: CoAP Message Format as Defined in RFC 7252

The message format for CoAP over TCP is very similar to the format specified for CoAP over UDP. The differences are as follows:

- Since the underlying TCP connection provides retransmissions and deduplication, there is no need for the reliability mechanisms provided by CoAP over UDP. The Type (T) and Message ID fields in the CoAP message header are elided.

- The Version (Vers) field is elided as well. In contrast to the message format of CoAP over UDP, the message format for CoAP over TCP includes only the fields that are necessary for the exchange of message content.
The FEC type for the P2MP PW Upstream FEC Element is encoded as follows:

```
0 1 2 3
0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5
P2MP PW Up=0x82C | PW Type | PW Info Length |
| AGI Type | AGI Length | AGI Value |
~ AGI Value (contd.) ~
| AII Type | AII Length | AII Value |
~ AII Value (contd.) ~
| PMSI Tunnel Typ| PMSI TT Length |
```

Figure 2: P2MP PW Upstream FEC Element

* P2MP PW Up:

8-bit representation for the P2MP PW Upstream FEC type.

* C bit:

A value of 1 or 0 indicating whether a control word is present or absent for the P2MP PW.

The Relay Source Port Option for DHCPv6

Relay Source Port Option" is a new DHCPv6 option. It MUST be either:

1) a DHCPv6 relay agent that uses a non-DHCP UDP port communicating with the IPv6 server and the upstream relay
2) an IPv6 relay agent that detects the use of a non-DHCP (not 547) by a downstream relay agent.

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

```
<table>
<thead>
<tr>
<th>Ver</th>
<th>T</th>
<th>TKL</th>
<th>Code</th>
<th>Message ID</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 2 3</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
```

3.1. Availability CSI-TLV

This option is defined in [RFC8258]. This document defines a new CSI-TLV called the Availability CSI-TLV that can be included one or more times.

3.2. Message Format

As defined in [RFC7252], as shown in Figure 3, CoAP over TCP uses the same transport (UDP, or DTLS over UDP) for keeping state and for providing length

```
<table>
<thead>
<tr>
<th>Ver</th>
<th>T</th>
<th>TKL</th>
<th>Code</th>
<th>Message ID</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 2 3</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
```

Figure 3: CoAP Message Format as Defined in RFC 7252

The message format for CoAP over TCP is very similar to the format specified for CoAP over UDP. The differences are as follows:

* Since the underlying TCP connection provides retransmissions and deduplication, there is no need for the reliability mechanisms provided by CoAP over UDP. The Type (T) and Message ID fields in the CoAP message header are elided.

* The Version (Vers) field is elided as well. In contrast to the message format of CoAP over UDP, the message format for CoAP over

Many variations with subtle differences → difficult to parse
Augmented ASCII diagrams

• Much can be achieved just by being consistent

• Need other elements: constraints on field values, optional fields, links between PDUs, …

• Adheres to the design principles given earlier
Parsing protocol standards

- Parse input into an RFC document object model
- RFC DOM is already well specified
- Allows for different input formats
Parses protocol standards

- Extract a protocol definition from the RFC DOM, and capture it in an intermediate representation

- Captures the syntax of the protocol and how to parse it

- Allows for different input languages, whose expressivity might vary
Intermediate representation captures all of metadata required to parse the protocol

• The layout of each PDU

• Parsing context for out-of-band data

• Helper methods for encrypted fields
Parsing protocol standards

- Generate parser code from the intermediate representation

- Split means that a parser generator only needs to be written once per output language
Summary

- IETF standardisation process can create ambiguous standards: want to introduce tooling without harming the parts of the process that work well

- ASCII diagrams already capture much of a protocol’s syntax

- Augmenting ASCII diagrams and using them consistently allows tooling to extract protocol syntax

- Capturing protocol parsing in a common intermediary format allows for flexibility

- Automated parser generation from the intermediary format enables test-driven development → better standards