Parsing Protocol Standards,
Parsing Standard Protocols

Colin Perkins
How do we specify network protocol standards?

Can we improve the robustness of the network, by improving the way we develop such standards?
Protocol Standards

- Internet is a network of networks
- Each runs a complex mixture of standard and proprietary protocols, implemented by the lowest bidder
- If this is to work, standards must be clearly written and straightforward to implement – are they?
Standards Development

- Open Systems Interconnection Reference Model
  - Textbook description of network standards
  - Cleanly structured, well organised
  - Neat
  - Logical
Standards Development

You are Here

Political
Financial
Application
Presentation
Session
Transport
Network
Link
Physical
Standards Development
Standards Development

- Standards are about people; people are political; agreements are hard fought
- “We reject: kings, presidents and voting. We believe in: rough consensus and running code.” – David D. Clark
What’s Wrong With Protocol Standards?

• The process works – the Internet has been a success!
• But… problems with the standards follow from the process:
  • Standards are developed incrementally by large teams
  • Participants in the process have conflicting goals
  • Takes months, perhaps years, to produce a specification
  • Resulting standards are primarily English prose
  • Resulting protocols are often found to contain inconsistencies, ambiguities, and vulnerabilities – due to the process, and because there’s no automated way of validating them for correctness
• This is not the fault of the standards setters – the process dooms them to fail
The process *is* inherently political and takes time; attempts to avoid that will fail.

Must work within existing standards development process; that’s where we find relevant stakeholders.

Desirable to increase testability and simplify implementation – help, not replace, the standards process.

---

How Can We Improve Standard Protocols?
Consensus Standards Take Time

- It takes time to forge consensus
- Standards cannot be written in a single sitting
  - Successful standards evolve over time
  - Written and extended over many years by a large and changing team
- Specifications are immutable once published – can work around bugs and misinterpretations, but cannot fix them
  - The deployed base overrides anything written in the standard
Work With Existing Stakeholders

- Work with researchers, engineers, and engineering managers
- Overwhelmingly systems focussed, experienced, conservative
- They write C code and care about bits – or, they manage those who do
Improve the Process – Don’t Try to Replace It

• Augment existing processes – don’t try to replace them
  • Organisational inertia is high
  • What we have may be sub-optimal, but *it works*

• Goal: to increase testability of protocol standards
  • Extract features from draft standards
  • Tools to validate protocol behaviour
  • Find bugs early – not as result of PhD students running formal analysis tools on published standards
  • Get standards community used to running automated tooling on drafts as they’re developed – introduce formalisms by the back door
Make the idea of automated protocol testing and analysis a normal part of the standards process
Where to Start?

• Parsing
• State Machines
• Two fundamental components of network protocols
Parsing Protocol Data

- Start with the basics:
  - Build tooling to check that protocol data units can be parsed
  - Enable automatic generation of parsing/serialisation code

- If we can’t even generate parsers for protocol data units, how can we check anything else?
But, isn’t parsing a solved problem…?
Practical Parsing Concerns

• Packet formats described informally, in English prose – not intended to be machine parsable
  • No consistency between specifications
  • Parsing relies on out-of-band knowledge
  • Data formats are increasingly encrypted
Some formats might be machine parsable...

...others seem almost intentionally written in a style that precludes it!
Many packet description formats, varying functionality and formality of specification – no real consistency
Version Negotiation packet

The version negotiation packet format is:

- Header Form: 1 bit (set to 1)
- Unused: 7 bits
- Version: 32 bits
- Destination Connection ID Length: 4 bits
- Source Connection ID Length: 4 bits
- Destination Connection ID: 0 or 32 to 144 bits
- Source Connection ID: 0 or 32 to 144 bits
- Supported Versions: 1 or more 32 bit fields

Notes:

- This is not the long header format, but it does have the same value for the first bit. Therefore, this bit alone is not sufficient to determine how the packet should be parsed: the Version field must be set to 0.

2.7 RTP

Increasing reliance on contextual metadata to describe packet formats

- Formats are not self-describing
  - HTTP parseable without extra information
  - RTP needs signalling to describe header and payload format
4.2. Short Header

<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>
<tr>
<td>+---------------+---------------+---------------</td>
<td>Destination Connection ID (0..144)</td>
<td>...</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>+---------------+---------------+---------------</td>
<td>Packet Number (8/16/32)</td>
<td>...</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>+---------------+---------------+---------------</td>
<td>Protected Payload (*)</td>
<td>...</td>
<td></td>
<td></td>
<td></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>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Figure 2: Short Header Packet Format

The short header can be used after the version and 1-Rtt keys are negotiated. Packets that use the short header contain the following fields:

Header Form: The most significant bit (0x80) of octet 0 is set to 0 for the short header.

Key Phase Bit: The second bit (0x40) of octet 0 indicates the key phase, which allows a recipient of a packet to identify the packet protection keys that are used to protect the packet. See [QUIC-TLS] for details.

Third Bit: The third bit (0x20) of octet 0 is set to 1.

Fourth Bit: The fourth bit (0x10) of octet 0 is set to 1.

[[Editor's Note: this section should be removed and the bit definitions changed before this draft goes to the IESG.]]

• Packet formats increasing comprised of structured fields to be parsed – but must be decrypted first
  • How is keying specified?
  • How to specify field transformations?
Representing Parsers

• How to build tooling to generate parsers from protocol standards, while working within the existing process?
Representing Protocol Parsers

- Where possible, re-use existing input formats
- If not possible, define formalised variants that are machine parsable, and are as close to the original as possible – and build tools to check syntax

- Goal: close compatibility with existing practice
- Different input formats for different use cases
Representing Protocol Parsers

- Define single intermediate format, to represent the possible input formats
- Internal to the tooling – can be as expressive as we want, since not visible to standards setters
Representing Protocol Parsers

- Generate implementations from intermediate format
Intermediate Parser Format

- Base type: fixed-width bit string
  - Bit width may be left unspecified
- Additional types:
  - Array – length may be unspecified
  - Structure type
  - Enumerated type – list of alternatives
  - Derived types – new type name
- External functions
  - Function signatures specified; body defined in the prose of the specification
- Parsing context
  - Used to hold persistent state, possibly defined out-of-band – e.g., decryption keying material
Intermediate Parser Format: Field Types

<table>
<thead>
<tr>
<th>Type</th>
<th>Data Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bit</td>
<td>BitString1</td>
</tr>
<tr>
<td>Version</td>
<td>BitString2</td>
</tr>
<tr>
<td>CsrcLength</td>
<td>BitString4</td>
</tr>
<tr>
<td>PayloadType</td>
<td>BitString8</td>
</tr>
<tr>
<td>SeqNum</td>
<td>BitString16</td>
</tr>
<tr>
<td>Timestamp</td>
<td>BitString32</td>
</tr>
<tr>
<td>SSRC</td>
<td>BitString32</td>
</tr>
<tr>
<td>PaddingLength</td>
<td>BitString8</td>
</tr>
</tbody>
</table>

Version Negotiation packet

The version negotiation packet format is:

- **Header Form**: 1 bit (set to 1)
- **Unused**: 7 bits
- **Version**: 32 bits
- **Destination Connection ID Length**: 4 bits
- **Source Connection ID Length**: 4 bits
- **Destination Connection ID**: 0 or 32 to 144 bits
- **Source Connection ID**: 0 or 32 to 144 bits
- **Supported Versions**: 1 or more 32 bit fields

Notes:

- This is not the long header format, but it does have the same value for the first bit. Therefore, this bit alone is not sufficient to determine how the packet should be parsed: the Version field must be set to 0.

2.7 RTP

```
+-------------+-------------+-------------+-------------+-------------+-------------+-------------+-------------+-------------+-------------+-------------+-------------+-------------+-------------+
|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) |
| |Padding (PadCnt octets, if P=1)|PadCnt (if P=1) |
+-------------+-------------+-------------+-------------+-------------+-------------+-------------+-------------+-------------+-------------+-------------+-------------+-------------+
Intermediate Parser Format: Structure Types

```c
struct HeaderExt = {
  id : ExtnType;
  len : ExtnSize;
  payload : Bit[];
} where {
  payload.length == 32*len;
};

struct RTPPacket = {
  version           : Version;
  padding_flag      : Bit;
  extension         : Bit;
  csrc_count        : CsrcLength;
  marker            : Bit;
  payload_type      : PayloadType;
  sequence_num      : SeqNum;
  timestamp         : Timestamp;
  ssrc_id           : SSRC;
  csrc_list         : SSRC[];
  header_extension  : HeaderExt if extension == 1
  protected_payload : decrypt(Bit[]) => payload : Bit[]
  padding           : Bit[];
  padding_count     : PaddingLength;
} where {
  version == 2;
  csrc_list.length == 32*4*crsc_count;
  padding.length == ((padding_count) ? 8*padding_count : 0);
};
```

Optional fields, transformed fields, and constraints

---

Version Negotiation packet

The version negotiation packet format is:

- **Header Form**: 1 bit (set to 1)
- **Unused**: 7 bits
- **Version**: 32 bits
- **Destination Connection ID Length**: 4 bits
- **Source Connection ID Length**: 4 bits
- **Destination Connection ID**: 0 or 32 to 144 bits
- **Source Connection ID**: 0 or 32 to 144 bits
- **Supported Versions**: 1 or more 32 bit fields

Notes:

- This is not the long header format, but it does have the same value for the first bit. Therefore, this bit alone is not sufficient to determine how the packet should be parsed: the Version field must be set to 0.

---

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

+---+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|V=2|P|X| CC |M| PT | sequence number |
+---+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
  | timestamp |
+---+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
  | synchr
  | o
  | on
  | i
  | z
  | a
  | t
  | i
  | o
  | n
  | s
  | e
  | n
  | s
  | o
  | r
  | c
  | e
  | n
  | t
  | i
  | d
  | (SSRC) identifier
  | (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) |
+---+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
  | Padding (PadCnt octets, if P=1)|PadCnt (if P=1) |
  | +-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
  | | ID | L=0 | data | ID |L=1| data ...
  | +-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
  | ...data | 0 (pad) | 0 (pad) | ID |L=3 |
  | +-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
  | ...data |
  | +-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
```

---

Optional fields, transformed fields, and constraints
Representing Protocol Parsers

• Intermediate format sufficient to describe IETF standard protocols
  • Requires complexity of structure definitions – optional fields, constraints, and transformations
  • Requires parsing context, to save state between packets or from signalling
  • Requires external functions – delegate functionality to yet-to-be-specified function

• Baseline implementation exists
  • Simple input format (like TLS representation language) → C-based parser
  • Parser for structured ASCII packet diagrams, and Rust-based backend, in development
  • Talk to me offline if interested in further details
Summary/Next Steps

• Protocol parsing is difficult, since specifications written in English
  • Common intermediate format, multiple front- and back-ends → feasible parser
  • Developing more structured versions of ASCII packet diagrams; integrating with IETF datatracker; encourage structured specifications

• Next steps – structured English for partial state machine specification?

• Challenge is to change processes in social organisations – research must integrate existing processes
  • We have excellent tools for protocol analysis – that aren’t widely used
  • Tooling suit the community