Tools for Disambiguating RFCs

Jane Yen, Barath Raghavan, Ramesh Govindan

USC Viterbi
School of Engineering
RFC production

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>RFC COUNT</td>
<td>310</td>
<td>263</td>
<td>208</td>
<td>180</td>
<td>209</td>
<td>240</td>
</tr>
</tbody>
</table>
We need to standardize this XYZ protocol.

XYZ protocol uses three way handshake…

Working Group

Communication

Specification Author
Ambiguities

Introduction

The Internet Protocol (IP) [1] is used for host-to-host datagram service in a system of interconnected networks called the Catenet [2]. The network connecting devices are called Gateways. These gateways communicate between themselves for control purposes via a Gateway to Gateway Protocol (GCP) [3,4]. Occasionally a gateway or destination host will communicate with a source host, for example, to report an error in datagram processing. For such purposes this protocol, the Internet Control Message Protocol (ICMP), is used. ICMP uses the basic support of IP as if it were a higher level protocol, however, ICMP is actually an integral part of IP, and must be implemented by every IP module.

ICMP messages are sent in several situations: for example, when a
Classic examples

The checksum is the 16-bit one’s complement of the one’s complement sum of the ICMP message starting with the ICMP Type.
Classic examples

The checksum is the 16-bit one’s complement of the one’s complement sum of the ICMP message starting with the ICMP Type.

Ending at ?
Classic examples

The checksum is the 16-bit one’s complement of the one’s complement sum of the ICMP message starting with the ICMP Type.

Ending at ?

Checksum the header OR Checksum both the header and payload
Possible Consequences

- Buggy Implementation
- Security Vulnerability
Rigorous discussion

This sentence is ambiguous. Please rephrase.

The $BOO$ field **MUST** be 0

Working Group

Communication * N

Specification Author
Are we close to near 0-ambiguity specification?
Uncover 5 instances of ambiguity and 6 instances of under-specification in ICMP RFC

Generate executable code of unambiguous specification that interoperate with 3rd party code

Generalize to sections of BFD, IGMP and NTP
Our Approach: Use Natural Language Processing

Natural language processing (NLP) on English specifications

Semantic parsing:
- Understand the semantics of a specification

Introduction
The Internet Protocol (IP) [1] is used for host-to-host datagram service in a system of interconnected networks called the Catenet [2]. The network connecting devices are called Gateways. These gateways communicate between themselves for control purposes via a Gateway to Gateway Protocol (GCP) [3,4]. Occasionally a gateway or destination host will communicate with a source host, for example, to report an error in datagram processing. For such purposes this protocol, the Internet Control Message Protocol (ICMP), is used. ICMP, uses the basic support of IP as if it were a higher level protocol, however, ICMP is actually an integral part of IP, and must be implemented by every IP module.

ICMP messages are sent in several situations: for example, when a
Goal

Executable protocol code generation

Specification

Ambiguity Discovery

Executable Code

void fill_icmp_echo_sender(Echo_or_Echo_Reply_Message_hdr *hdr, uint16_t length, int type_value) {
    char *data = (char *) (hdr + 1);
    // 8 for echo message,
    // 0 for echo reply message
    hdr->type = type_value;
    // Set code to 0
    hdr->code = 0;
    // If code equals 0, an identifier may be zero to help match echos and replies
    if (hdr->code == 0) {
        hdr->identifier = 0;
        // If code equals 0, a sequence number may be zero to help match echos and replies
        if (hdr->code == 0) {
            hdr->sequence_number = 0;
            // For computing the checksum, the checksum field should be zero
            hdr->checksum = 0;
            // For computing the checksum, if the total length is odd, the received data
            // is padded with one octet of zeros
            if (isodd(length)) {
                pad(data, sizeof(*data), 0, 1);
                length += 1;
            }
            // The checksum is the 16-bit one’s complement of the one’s complement sum of
            // the ICMP message starting with the ICMP Type
            hdr->checksum = us16bit_ones_complement(
                ...
Human in the Loop

Semantic parsing is not perfect
Challenges
Challenges

- Specifications use domain-specific language

- 

- 

-
Challenges

- Specifications use domain-specific language
- Semantic parsers have limitations
Challenges

- Specifications use domain-specific language
- Semantic parsers have limitations
- Semantic representations need to be converted into code
Contributions

- Specifications use domain-specific language
  
  *Extend semantic parser with domain-specific syntax and semantics*

- Semantic parsers have limitations

- Semantic representations need to be converted into code
Contributions

● Specifications use domain-specific language

  Extend semantic parser with domain-specific syntax and semantics

● Semantic parsers have limitations

  Automate disambiguation of poor semantic representations with checking rules

● Semantic representations need to be converted into code
Contributions

● Specifications use domain-specific language
  Extend semantic parser with domain-specific syntax and semantics

● Semantic parsers have limitations
  Automate disambiguation of poor semantic representations with checking rules

● Semantic representations need to be converted into code
  Compile semantic representations into executable code
Sage Components

Semantic Parsing → Disambiguation → Code Generator
Sage Workflow
Sage Workflow

RFC → Semantic Parsing → Disambiguation → Code Generator

More than 1 representations

Rewrite sentences
Sage Workflow
Sage Components

Semantic Parsing → Disambiguation → Code Generator
Semantic parsing

**Input**
Sentence: “Checksum is zero”

**NLP Parser**

**Output**
Logical Form: @Is(“checksum”, 0)
Key Observation

A logical form is a unifying abstraction for disambiguation and code generation
Domain Specific Extensions

Term dictionary with generic noun or noun phrase labeler

- 
- 
- 

Domain specific semantics

- 
- 

Domain Specific Extensions

Term dictionary with generic noun or noun phrase labeler

- Part of speech tagging: SpaCy
- Extended SpaCy's term dictionary
  - e.g., “one’s complement”

Domain specific semantics

- 
  - 
  - 

Domain Specific Extensions

Term dictionary with generic noun or noun phrase labeler

- Part of speech tagging: SpaCy
- Extended SpaCy's term dictionary
  - e.g., “one’s complement”

Domain specific semantics

- Idiomatic usage
  - e.g., “=” sign in “0 = Echo Reply”
Sage Components

Semantic Parsing → Disambiguation → Code Generator
Ambiguity

CCG parser could generate **zero or more than one** logical forms (LFs)
Ambiguity

CCG parser could generate **zero or more than one** logical forms (LFs)

| Incomplete | If code = 0, identifies the octet where an error was detected | 0 LF |
## Ambiguity

CCG parser could generate **zero or more than one** logical forms (LFs)

<table>
<thead>
<tr>
<th>Incomplete</th>
<th>If code = 0, identifies the octet where an error was detected</th>
<th>0 LF</th>
</tr>
</thead>
<tbody>
<tr>
<td>Imprecise language</td>
<td>To form an information reply message, the source and destination addresses are simply reversed, the <strong>type code</strong> changed to 16, and the checksum recomputed</td>
<td>code? type?</td>
</tr>
</tbody>
</table>
### Winnowing Ambiguous Logical Forms

<table>
<thead>
<tr>
<th>Type</th>
<th>Checking rules</th>
</tr>
</thead>
<tbody>
<tr>
<td>Argument ordering</td>
<td></td>
</tr>
<tr>
<td>Predicate ordering</td>
<td></td>
</tr>
<tr>
<td>Distributivity</td>
<td></td>
</tr>
<tr>
<td>Associativity</td>
<td></td>
</tr>
</tbody>
</table>

- **1+ LFs**
- **1 LF**
SAGE Components

Semantic Parsing → Disambiguation → Code Generator
Logical Forms to Code

Input: Logical Form: `@Is("checksum", 0)`

Contextual Information

Code generator

Output: `hdr->checksum = 0;`
## Evaluation

<table>
<thead>
<tr>
<th><strong>TEST COMMANDS</strong></th>
<th><strong>PURPOSE</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td>client ping <code>–c 10 10.0.1.1</code></td>
<td>Test echo msg</td>
</tr>
<tr>
<td>client ping <code>–c 10 192.168.3.1</code></td>
<td>Test dest unreachable msg</td>
</tr>
<tr>
<td>client ping <code>–c 10 –t 1 192.168.2.2</code></td>
<td>Test time exceeded msg</td>
</tr>
<tr>
<td>client traceroute 10.0.1.1</td>
<td>Test traceroute</td>
</tr>
</tbody>
</table>

![Graph showing the number of logical forms for different parameters](image)
Beyond Sage, there remains many challenges unaddressed.
SAGE Limitations

- Specification components

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>◆ Packet Format</td>
<td>Packet anatomy (i.e., field structure)</td>
</tr>
<tr>
<td>◆ Field Descriptions</td>
<td>Packet header field descriptions</td>
</tr>
<tr>
<td>◆ Constraints</td>
<td>Constraints on field values</td>
</tr>
<tr>
<td>◆ Protocol Behaviors</td>
<td>Reactions to external/external events</td>
</tr>
<tr>
<td>System Architecture</td>
<td>Protocol implementation components</td>
</tr>
<tr>
<td>+ State Management</td>
<td>Session information and/or status</td>
</tr>
<tr>
<td>Comm. Patterns</td>
<td>Message sequences (e.g., handshakes)</td>
</tr>
</tbody>
</table>

Table 1: Protocol specification components. SAGE supports those marked with ◆ (fully) and + (partially).
Challenges

- Paragraph or sentence-based analysis
- Mis-matched/mis-captured behaviors
- Standalone or multiple RFCs
- Single protocol or stack of protocols
- Logic v.s. performance
Challenges

- Paragraph or sentence-based analysis
- Mis-matched/mis-captured behaviors
- Standalone or multiple RFCs
- Single protocol or stack of protocols
- Logic v.s. performance
Challenges

- Paragraph or sentence-based analysis
- Mis-matched/mis-captured behaviors
- Standalone or multiple RFCs
- Single protocol or stack of protocols
- Logic v.s. performance
Challenges

- Paragraph or sentence-based analysis
- Mis-matched/mis-captured behaviors
- Standalone or multiple RFCs
- Single protocol or stack of protocols
- Logic v.s. performance
Challenges

- Paragraph or sentence-based analysis
- Mis-matched/mis-captured behaviors
- Standalone or multiple RFCs
- Single protocol or stack of protocols
- Logic v.s. performance
Current Work
Reduce Human Effort

- RFC
- Semantic Parsing
- Disambiguation
- Code Generator
- Code

Rewrite sentences

More than 1 representations
Failing unit tests
Challenges

- Can we avoid writing an ambiguous sentence in the first place?

- What kind of protocols are we going to support?
Solution Directions

- Can we avoid writing an ambiguous sentence in the first place?

  A user interface guides spec author to produce only essential information

- What kind of protocols are we going to support?
Solution Directions

- Can we avoid writing an ambiguous sentence in the first place?

A user interface guides spec author to write unambiguous sentences

- What kind of protocols are we going to support?

Stateful protocols
(It requires to keep internal states to decide operations)

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>✤ Packet Format</td>
<td>Packet anatomy (i.e., field structure)</td>
</tr>
<tr>
<td>✤ Field Descriptions</td>
<td>Packet header field descriptions</td>
</tr>
<tr>
<td>✤ Constraints</td>
<td>Constraints on field values</td>
</tr>
<tr>
<td>✤ Protocol Behaviors</td>
<td>Reactions to external/internal events</td>
</tr>
<tr>
<td>✤ System Architecture</td>
<td>Protocol implementation components</td>
</tr>
<tr>
<td>✤ State Management</td>
<td>Session information and/or status</td>
</tr>
<tr>
<td>✤ Comm. Patterns</td>
<td>Message sequences (e.g., handshakes)</td>
</tr>
</tbody>
</table>

Table 1: Protocol specification components. SAGE supports those marked with ✤ (fully) and + (partially).
Our vision

User interface

Essential protocol elements
Our vision

User interface

Essential protocol elements

Executable code

English RFC
Our vision

User interface

Timer
Last received packet
Output packet
Program states

Essential protocol elements

English RFC

Executable code