Tools for Disambiguating RFCs

Jane Yen, Barath Raghavan, Ramesh Govindan

USC Viterbi
School of Engineering

RFC production

YEAR	2016	2017	2018	2019	2020	2021
RFC COUNT	310	263	208	180	209	240

Specification production

We need to standardize this XYZ protocol.

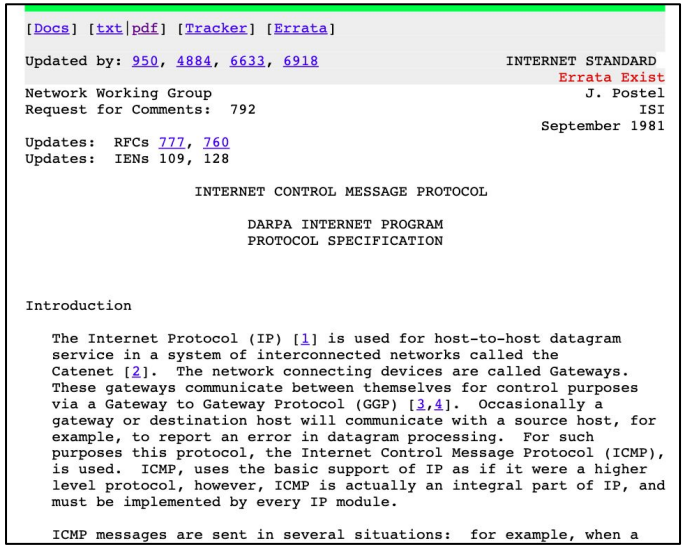
XYZ protocol uses three way handshake…

Communication

Working Group

Specification Author

Ambiguities



[Docs] [txt|pdf] [Tracker] [Errata]

Updated by: 950, 4884, 6633, 6918 — INTERNET STANDARD
Errata Exist

Network Working Group — J. Postel
Request for Comments: 792 — ISI
September 1981

Updates: RFCs 777, 760
Updates: IENs 109, 128

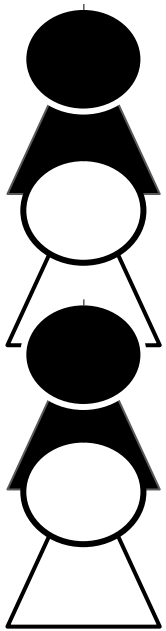
INTERNET CONTROL MESSAGE PROTOCOL

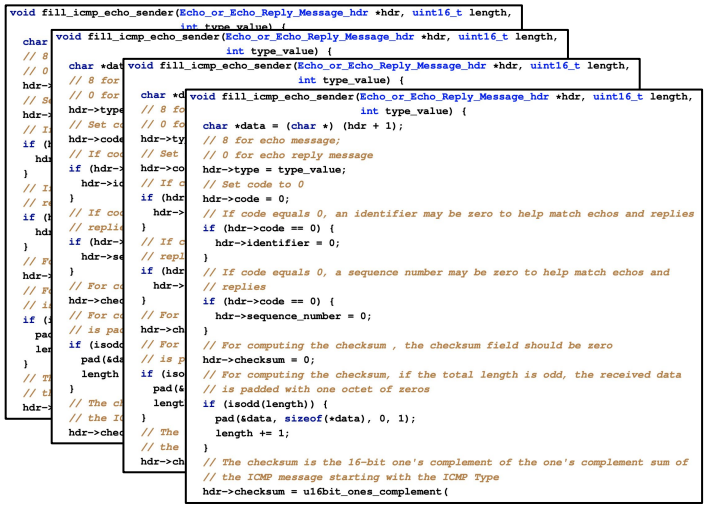
DARPA INTERNET PROGRAM
PROTOCOL SPECIFICATION

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 (GGP) [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





```
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 = u16bit_ones_complement(
```

Ambiguous Specification

Human Interpretation

Multiple versions

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

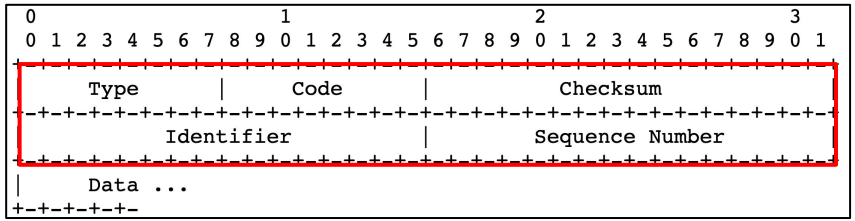
The checksum is the 16-bit one's complement of the one's complement sum of the ICMP message starting with the ICMP Type.

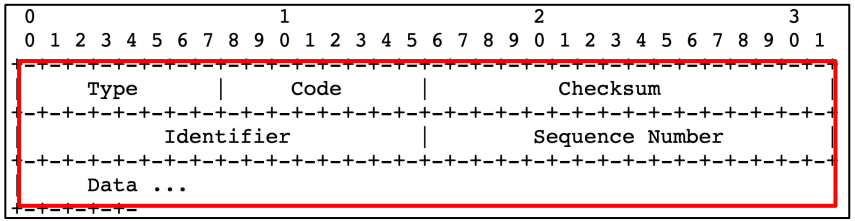
Ending at ?

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.

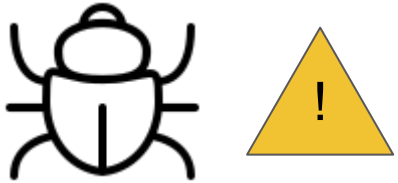
Ending at ?





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

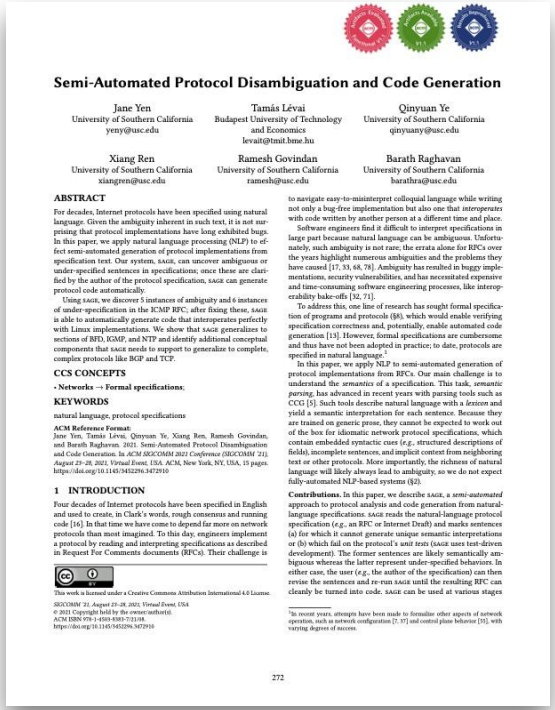
Communication * N

Working Group

Specification Author

Are we close to near 0-ambiguity specification?

Our work



Semi-Automated Protocol Disambiguation and Code Generation

Jane Yen
University of Southern California
yeny@usc.edu

Tamás Lévai
Budapest University of Technology and Economics
levait@tmit.bme.hu

Qinyuan Ye
University of Southern California
qinyuany@usc.edu

Xiang Ren
University of Southern California
xiangren@usc.edu

Ramesh Govindan
University of Southern California
rameshg@usc.edu

Barath Raghavan
University of Southern California
barathra@usc.edu

ABSTRACT

For decades, Internet protocols have been specified using natural language. Given the ambiguity inherent in such text, it is not surprising that protocol implementations have long exhibited bugs. In this paper, we apply natural language processing (NLP) to effect semi-automated generation of protocol implementations from specification text. Our system, SAGE, can uncover ambiguous or under-specified sentences in specifications; once these are clarified by the author of the protocol specification, SAGE can generate protocol code automatically.

Using SAGE, we discover 5 instances of ambiguity and 6 instances of under-specification in the ICMP RFC, after fixing these, SAGE is able to automatically generate code that interoperates perfectly with Linux implementations. We show that SAGE generalizes to sections of BFD, IGMP, and NTP and identify additional conceptual components that SAGE needs to support to generalize to complete, complex protocols like BGP and TCP.

CCS CONCEPTS

• **Networks** → **Formal specifications**;

KEYWORDS

natural language, protocol specifications

ACM Reference Format:
Jane Yen, Tamás Lévai, Qinyuan Ye, Xiang Ren, Ramesh Govindan, and Barath Raghavan. 2021. Semi-Automated Protocol Disambiguation and Code Generation. In *ACM SIGCOMM 2021 Conference (SIGCOMM '21), August 23–28, 2021, Virtual Event, USA.* ACM, New York, NY, USA, 15 pages. https://doi.org/10.1145/3452296.3472910

1 INTRODUCTION

Four decades of Internet protocols have been specified in English and used to create, in Clark's words, rough consensus and running code [16]. In that time we have come to depend far more on network protocols than most imagined. To this day, engineers implement a protocol by reading and interpreting specifications as described in Request For Comments documents (RFCs). Their challenge is to navigate easy-to-misinterpret colloquial language while writing not only a bug-free implementation but also one that *interoperates* with code written by another person at a different time and place.

Software engineers find it difficult to interpret specifications in large part because natural language can be ambiguous. Unfortunately, such ambiguity is not rare; the errata alone for RFCs over the years highlight numerous ambiguities and the problems they have caused [17, 33, 68, 78]. Ambiguity has resulted in buggy implementations, security vulnerabilities, and has necessitated expensive and time-consuming software engineering processes, like interoperability bake-offs [32, 71].

To address this, one line of research has sought formal specification of programs and protocols (§8), which would enable verifying specification correctness and, potentially, enable automated code generation [13]. However, formal specifications are cumbersome and thus have not been adopted in practice; to date, protocols are specified in natural language.[1]

In this paper, we apply NLP to semi-automated generation of protocol implementations from RFCs. Our main challenge is to understand the *semantics* of a specification. This task, *semantic parsing*, has advanced in recent years with parsing tools such as CCG [5]. Such tools describe natural language with a *lexicon* and yield a semantic interpretation for each sentence. Because they are trained on generic prose, they cannot be expected to work out of the box for idiomatic network protocol specifications, which contain embedded syntactic cues (e.g., structured descriptions of fields), incomplete sentences, and implicit context from neighboring text or other protocols. More importantly, the richness of natural language will likely always lead to ambiguity, so we do not expect fully-automated NLP-based systems (§2).

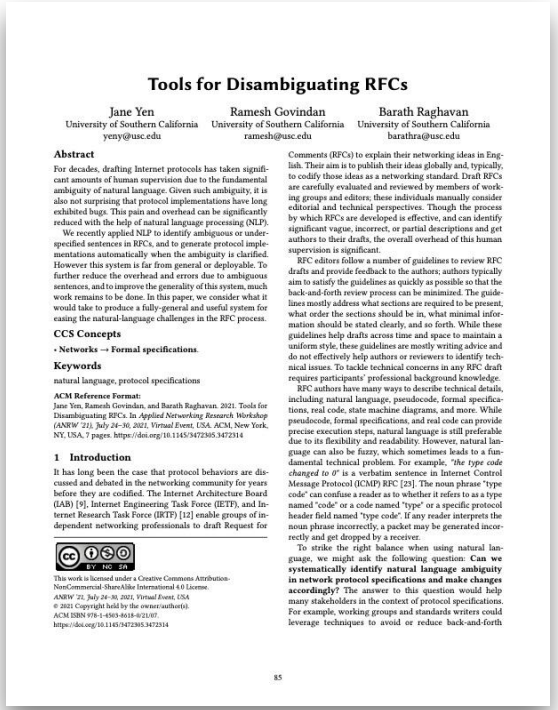
Contributions. In this paper, we describe SAGE, a *semi-automated* approach to protocol analysis and code generation from natural-language specifications. SAGE reads the natural-language protocol specification (e.g., an RFC or Internet Draft) and marks sentences (a) for which it cannot generate unique semantic interpretations or (b) which fail on the protocol's *unit tests* (SAGE uses test-driven development). The former sentences are likely semantically ambiguous whereas the latter represent under-specified behaviors. In either case, the user (e.g., the author of the specification) can then revise the sentences and re-run SAGE until the resulting RFC can cleanly be turned into code. SAGE can be used at various stages

[1] In recent years, attempts have been made to formalize other aspects of network operation, such as network configuration [7, 37] and control plane behavior [35], with varying degrees of success.

SIGCOMM '21, August 23–28, 2021, Virtual Event, USA

ACM ISBN 978-1-4503-8383-7/21/08.
https://doi.org/10.1145/3452296.3472910

SIGCOMM'21



Tools for Disambiguating RFCs

Jane Yen
University of Southern California
yeny@usc.edu

Ramesh Govindan
University of Southern California
rameshg@usc.edu

Barath Raghavan
University of Southern California
barathra@usc.edu

Abstract

For decades, drafting Internet protocols has taken significant amounts of human supervision due to the fundamental ambiguity of natural language. Given such ambiguity, it is also not surprising that protocol implementations have long exhibited bugs. This pain and overhead can be significantly reduced with the help of natural language processing (NLP).

We recently applied NLP to identify ambiguous or under-specified sentences in RFCs, and to generate protocol implementations automatically when the ambiguity is clarified. However this system is far from general or deployable. To further reduce the overhead and errors due to ambiguous sentences, and to improve the generality of this system, much work remains to be done. In this paper, we consider what it would take to produce a fully-general and useful system for easing the natural-language challenges in the RFC process.

CCS Concepts

• **Networks** → **Formal specifications**.

Keywords

natural language, protocol specifications

ACM Reference Format:
Jane Yen, Ramesh Govindan, and Barath Raghavan. 2021. Tools for Disambiguating RFCs. In *Applied Networking Research Workshop (ANRW '21), July 24–30, 2021, Virtual Event, USA.* ACM, New York, NY, USA, 7 pages. https://doi.org/10.1145/3472305.3472314

1 Introduction

It has long been the case that protocol behaviors are discussed and debated in the networking community for years before they are codified. The Internet Architecture Board (IAB) [9], Internet Engineering Task Force (IETF), and Internet Research Task Force (IRTF) [12] enable groups of independent networking professionals to draft Request for Comments (RFCs) to explain their networking ideas in English. Their aim is to publish their ideas globally and, typically, to codify those ideas as a networking standard. Draft RFCs are carefully evaluated and reviewed by members of working groups and editors; these individuals manually consider editorial and technical perspectives. Though the process by which RFCs are developed is effective, and can identify significant vague, incorrect, or partial descriptions and get authors to their drafts, the overall overhead of this human supervision is significant.

RFC editors follow a number of guidelines to review RFC drafts and provide feedback to the authors; authors typically aim to satisfy the guidelines as quickly as possible so that the back-and-forth review process can be minimized. The guidelines mostly address what sections are required to be present, what order the sections should be in, what minimal information should be stated clearly, and so forth. While these guidelines help drafts across time and space to maintain a uniform style, these guidelines are mostly writing advice and do not effectively help authors or reviewers to identify technical issues. To tackle technical concerns in any RFC draft requires participants' professional background knowledge.

RFC authors have many ways to describe technical details, including natural language, pseudocode, formal specifications, real code, state machine diagrams, and more. While pseudocode, formal specifications, and real code can provide precise execution steps, natural language is still preferable due to its flexibility and readability. However, natural language can also be fuzzy, which sometimes leads to a fundamental technical problem. For example, *"the type code changed to 0"* is a verbatim sentence in Internet Control Message Protocol (ICMP) RFC [23]. The noun phrase "type code" can confuse a reader as to whether it refers to as a type named "code" or a code named "type" or a specific protocol header field named "type code". If any reader interprets the noun phrase incorrectly, a packet may be generated incorrectly and get dropped by a receiver.

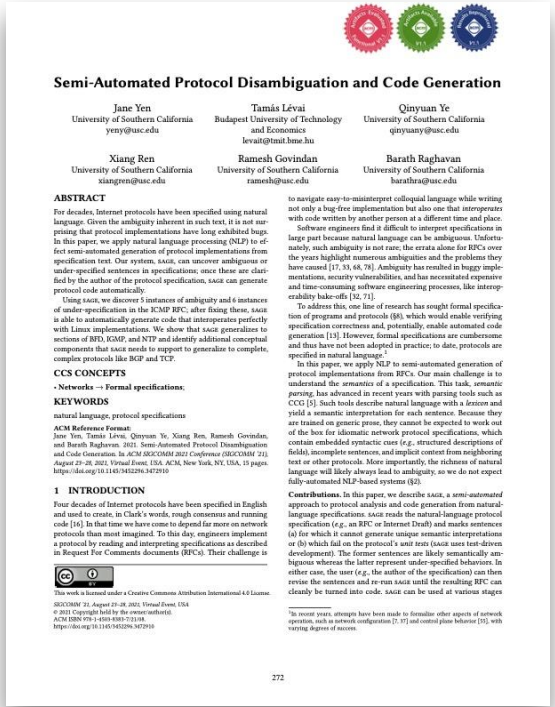
To strike the right balance when using natural language, we might ask the following question: **Can we systematically identify natural language ambiguity in network protocol specifications and make changes accordingly?** The answer to this question would help many stakeholders in the context of protocol specifications. For example, working groups and standards writers could leverage techniques to avoid or reduce back-and-forth

ANRW '21, July 24–30, 2021, Virtual Event, USA

ACM ISBN 978-1-4503-9618-0/21/07.
https://doi.org/10.1145/3472305.3472314

ANRW'21

Sage



Semi-Automated Protocol Disambiguation and Code Generation

Jane Yen
University of Southern California
yeny@usc.edu

Tamás Lévai
Budapest University of Technology and Economics
levait@tmit.bme.hu

Qinyuan Ye
University of Southern California
qinyuany@usc.edu

Xiang Ren
University of Southern California
xiangren@usc.edu

Ramesh Govindan
University of Southern California
ramesh@usc.edu

Barath Raghavan
University of Southern California
barathra@usc.edu

ABSTRACT

For decades, Internet protocols have been specified using natural language. Given the ambiguity inherent in such text, it is not surprising that protocol implementations have long exhibited bugs. In this paper, we apply natural language processing (NLP) to effect semi-automated generation of protocol implementations from specification text. Our system, SAGE, can uncover ambiguous or under-specified sentences in specifications; once these are clarified by the author of the protocol specification, SAGE can generate protocol code automatically.

Using SAGE, we discover 5 instances of ambiguity and 6 instances of under-specification in the ICMP RFC, after fixing these, SAGE is able to automatically generate code that interoperates perfectly with Linux implementations. We show that SAGE generalizes to sections of BFD, IGMP, and NTP and identify additional conceptual components that SAGE needs to support to generalize to complete, complex protocols like BGP and TCP.

CCS CONCEPTS

• **Networks** → **Formal specifications**;

KEYWORDS

natural language, protocol specifications

ACM Reference Format:

Jane Yen, Tamás Lévai, Qinyuan Ye, Xiang Ren, Ramesh Govindan, and Barath Raghavan. 2021. Semi-Automated Protocol Disambiguation and Code Generation. In *ACM SIGCOMM 2021 Conference (SIGCOMM '21), August 23–28, 2021, Virtual Event, USA.* ACM, New York, NY, USA, 15 pages. https://doi.org/10.1145/3452296.3472910

1 INTRODUCTION

Four decades of Internet protocols have been specified in English and used to create, in Clark's words, rough consensus and running code [16]. In that time we have come to depend far more on network protocols than most imagined. To this day, engineers implement a protocol by reading and interpreting specifications as described in Request For Comments documents (RFCs). Their challenge is to navigate easy-to-misinterpret colloquial language while writing not only a bug-free implementation but also one that *interoperates* with code written by another person at a different time and place.

Software engineers find it difficult to interpret specifications in large part because natural language can be ambiguous. Unfortunately, such ambiguity is not rare; the errata alone for RFCs over the years highlight numerous ambiguities and the problems they have caused [17, 33, 68, 78]. Ambiguity has resulted in buggy implementations, security vulnerabilities, and has necessitated expensive and time-consuming software engineering processes, like interoperability bake-offs [32, 71].

To address this, one line of research has sought formal specification of programs and protocols (§8), which would enable verifying specification correctness and, potentially, enable automated code generation [13]. However, formal specifications are cumbersome and thus have not been adopted in practice; to date, protocols are specified in natural language.[1]

In this paper, we apply NLP to semi-automated generation of protocol implementations from RFCs. Our main challenge is to understand the *semantics* of a specification. This task, *semantic parsing*, has advanced in recent years with parsing tools such as CCG [5]. Such tools describe natural language with a *lexicon* and yield a semantic interpretation for each sentence. Because they are trained on generic prose, they cannot be expected to work out of the box for idiomatic network protocol specifications, which contain embedded syntactic cues (*e.g.*, structured descriptions of fields), incomplete sentences, and implicit context from neighboring text or other protocols. More importantly, the richness of natural language will likely always lead to ambiguity, so we do not expect fully-automated NLP-based systems (§2).

Contributions. In this paper, we describe SAGE, a *semi-automated* approach to protocol analysis and code generation from natural-language specifications. SAGE reads the natural-language protocol specification (*e.g.*, an RFC or Internet Draft) and marks sentences (a) for which it cannot generate unique semantic interpretations or (b) which fail on the protocol's *unit tests* (SAGE uses test-driven development). The former sentences are likely semantically ambiguous whereas the latter represent under-specified behaviors. In either case, the user (*e.g.*, the author of the specification) can then revise the sentences and re-run SAGE until the resulting RFC can cleanly be turned into code. SAGE can be used at various stages

SIGCOMM '21, August 23–28, 2021, Virtual Event, USA

ACM ISBN 978-1-4503-8383-7/21/08.
https://doi.org/10.1145/3452296.3472910

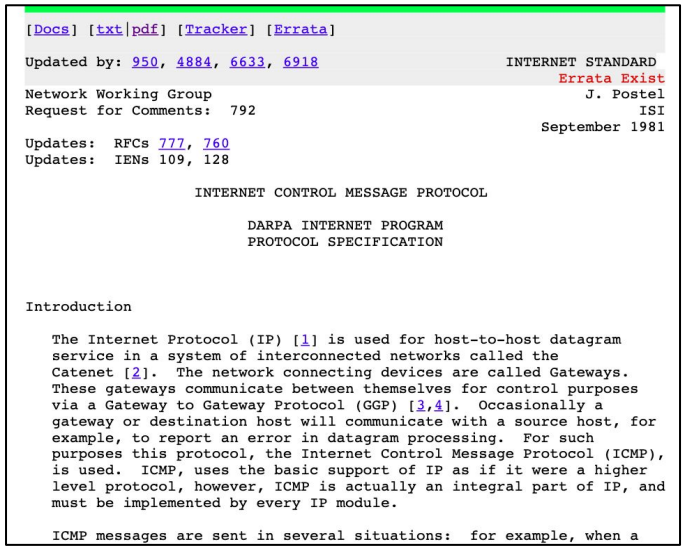
[1] In recent years, attempts have been made to formalize other aspects of network operation, such as network configuration [7, 37] and control plane behavior [35], with varying degrees of success.

SIGCOMM'21

- 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



[Docs] [txt|pdf] [Tracker] [Errata]

Updated by: 950, 4884, 6633, 6918 — INTERNET STANDARD — Errata Exist

Network Working Group — J. Postel
Request for Comments: 792 — ISI
September 1981

Updates: RFCs 777, 760
Updates: IENs 109, 128

INTERNET CONTROL MESSAGE PROTOCOL

DARPA INTERNET PROGRAM
PROTOCOL 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 (GGP) [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

Specification

NLP

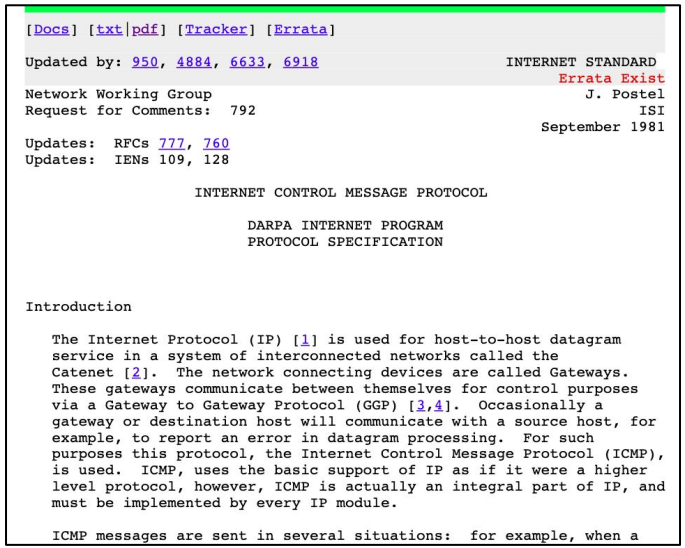
Ambiguity Discovery

Semantic parsing:

- Understand the semantics of a specification

Goal

Executable protocol code generation



[Docs] [txt|pdf] [Tracker] [Errata]

Updated by: 950, 4884, 6633, 6918 INTERNET STANDARD
Errata Exist

Network Working Group J. Postel
Request for Comments: 792 ISI
September 1981

Updates: RFCs 777, 760
Updates: IENs 109, 128

INTERNET CONTROL MESSAGE PROTOCOL

DARPA INTERNET PROGRAM
PROTOCOL SPECIFICATION

Introduction

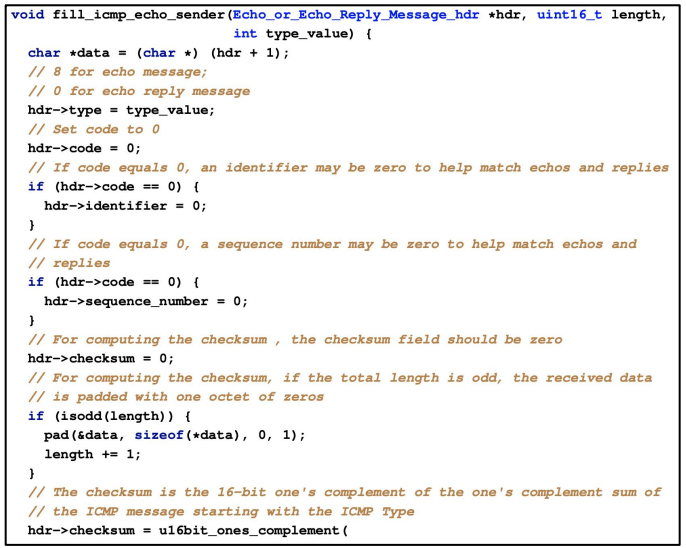
The Internet Protocol (IP) [1] is used for host-to-host datagram service in a system of interconnected networks called the Catenet [2]. The network connecting devices are called Gateways. These gateways communicate between themselves for control purposes via a Gateway to Gateway Protocol (GGP) [3,4]. 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

Specification

NLP

Ambiguity Discovery

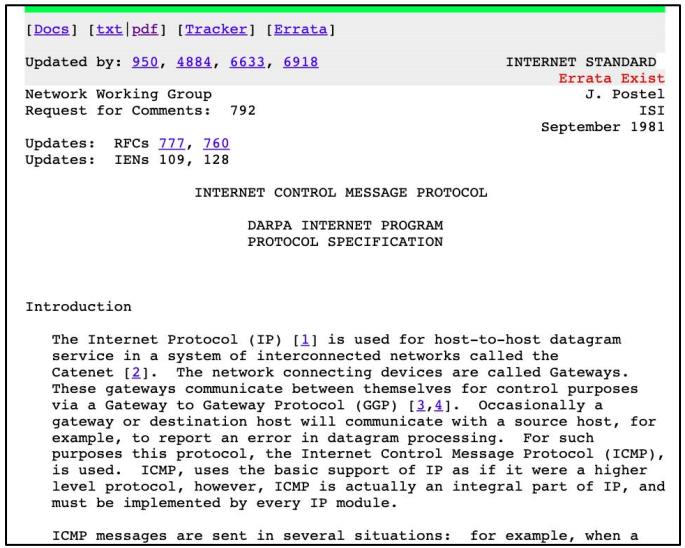


```
void fill_icmp_echo_sender(Echo_or_Echo_Reply_Message_hdr *hdr, uint16_t length,
                           int type_value) {
  char *data = (char *) (hdr + 1);
  // 8 for echo message;
  // 0 for echo reply message
  hdr->type = type_value;
  // Set code to 0
  hdr->code = 0;
  // If code equals 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 = u16bit_ones_complement(
```

Executable Code

Human in the Loop

Semantic parsing is not perfect



[Docs] [txt|pdf] [Tracker] [Errata]

Updated by: 950, 4884, 6633, 6918 INTERNET STANDARD
Errata Exist

Network Working Group J. Postel
Request for Comments: 792 ISI
September 1981

Updates: RFCs 777, 760
Updates: IENs 109, 128

INTERNET CONTROL MESSAGE PROTOCOL

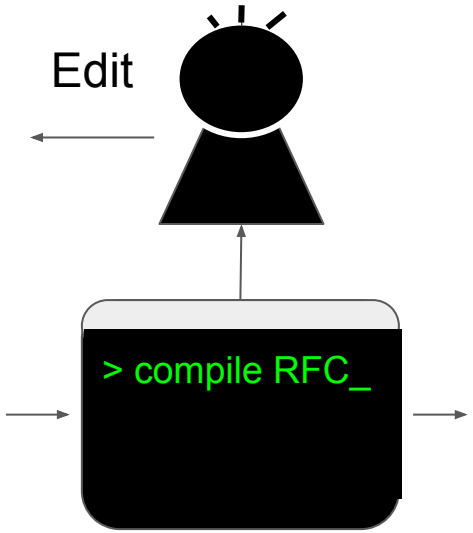
DARPA INTERNET PROGRAM
PROTOCOL SPECIFICATION

Introduction

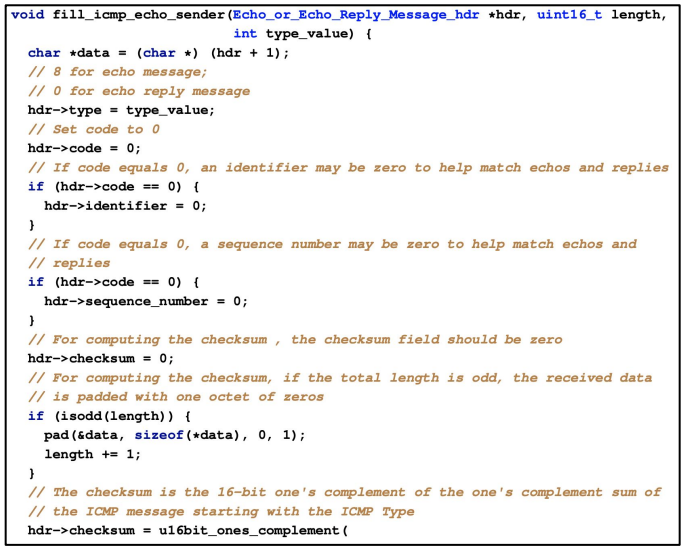
The Internet Protocol (IP) [1] is used for host-to-host datagram service in a system of interconnected networks called the Catenet [2]. The network connecting devices are called Gateways. These gateways communicate between themselves for control purposes via a Gateway to Gateway Protocol (GGP) [3,4]. 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

Specification



Ambiguity Discovery



```
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 = u16bit_ones_complement(
```

Executable Code

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

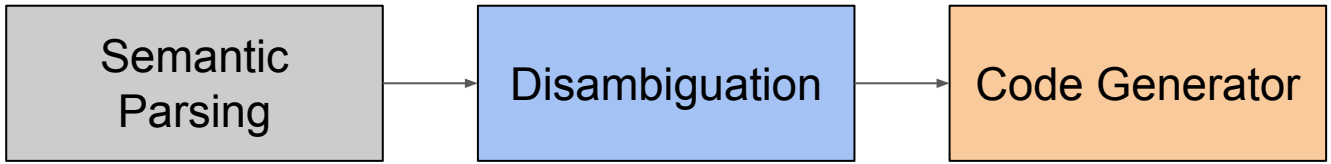
- Semantic parsers have limitations

  Automate disambiguation of poor semantic representations with checking rules

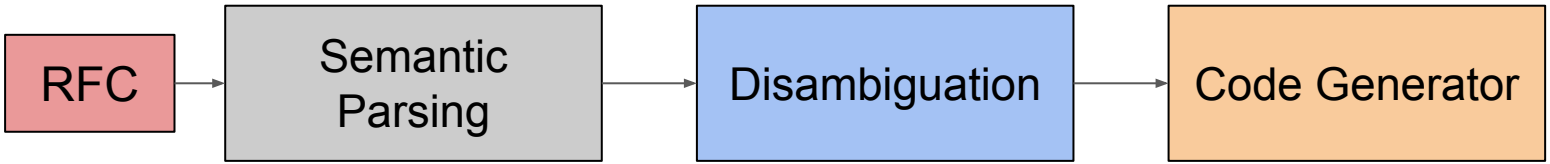
- Semantic representations need to be converted into code

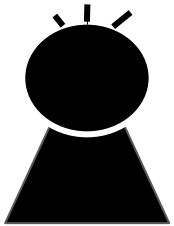
  Compile semantic representations into executable code

Sage Components

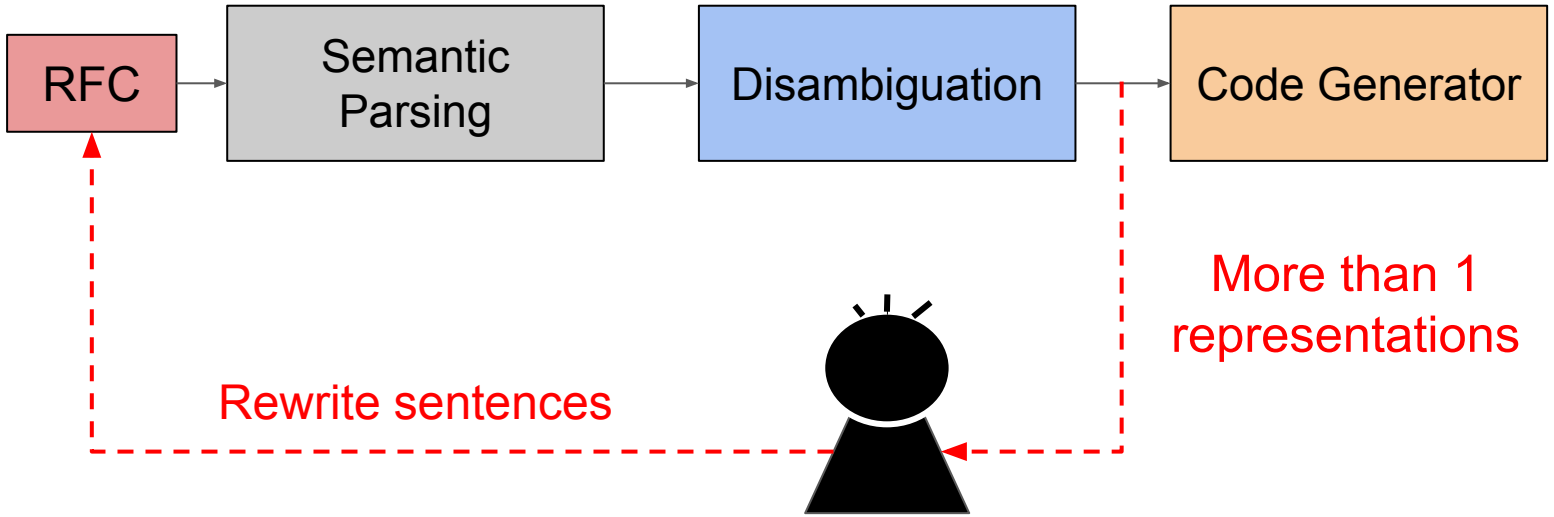


Sage Workflow

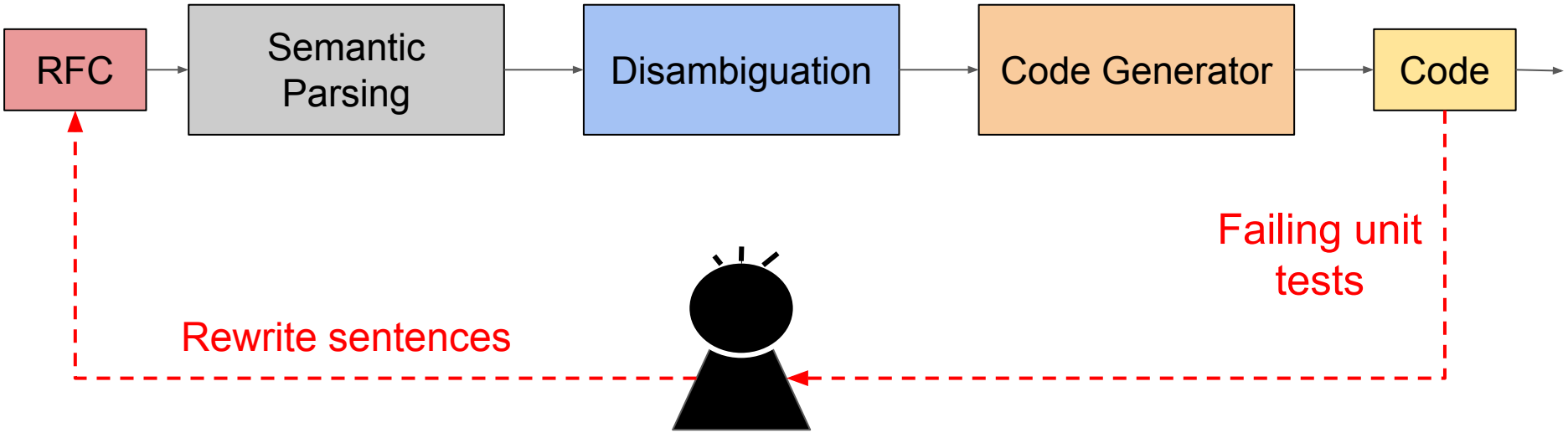




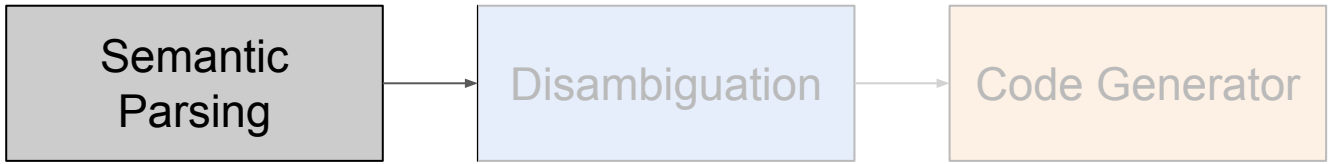
Sage Workflow



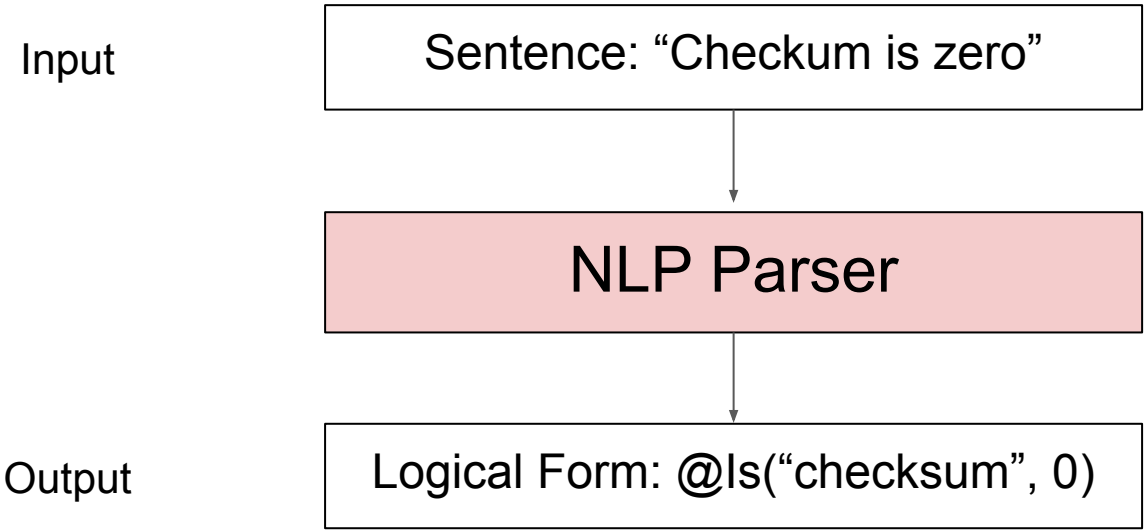
Sage Workflow



Sage Components



Semantic parsing



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

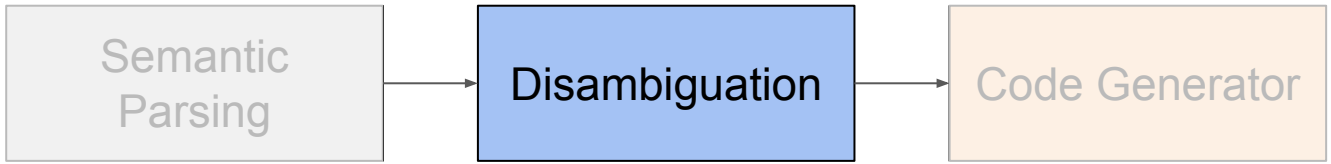
Term dictionary with generic noun or noun phrase labeler

- 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



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

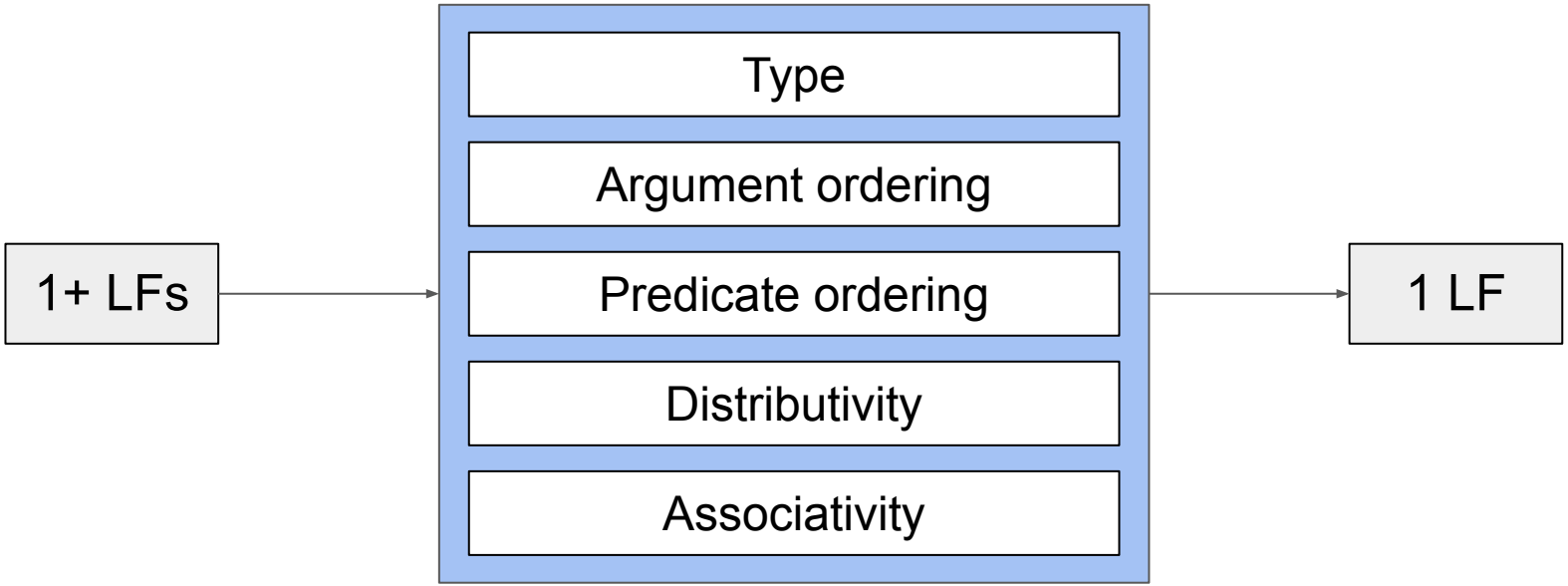
0 LF

Ambiguity

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

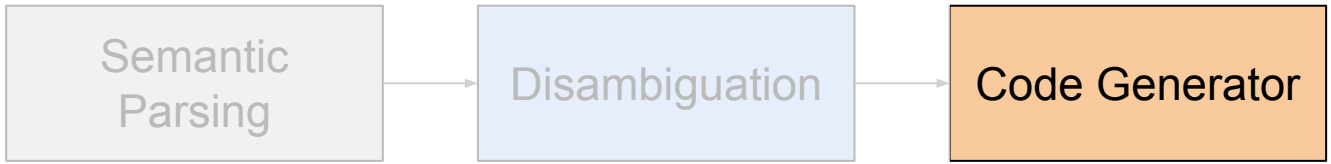
Incomplete	If code = 0, identifies the octet where an error was detected	0 LF
Imprecise language	To form a information reply message, the source and destination addresses are simply reversed, the type code changed to 16, and the checksum recomputed	code? type?

Winnowing Ambiguous Logical Forms

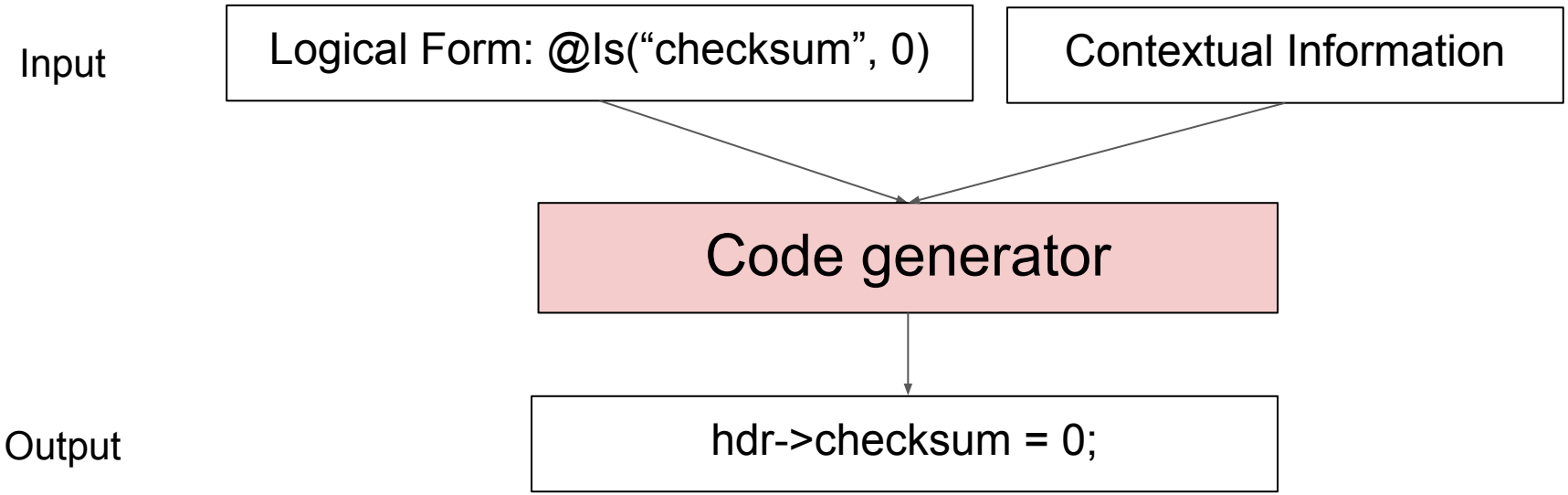


Checking rules

SAGE Components

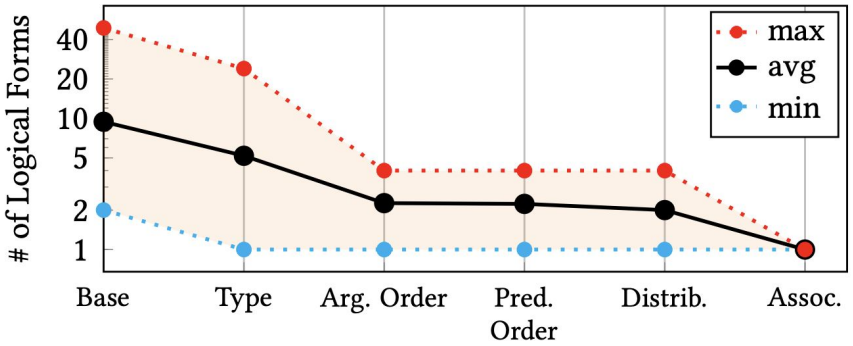


Logical Forms to Code



Evaluation

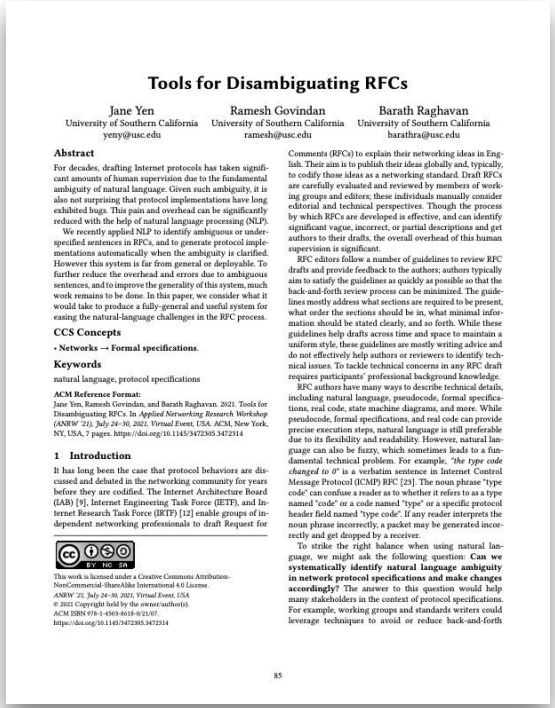
TEST COMMANDS	PURPOSE
`client ping -c 10 10.0.1.1`	Test echo msg
`client ping -c 10 192.168.3.1`	Test dest unreachable msg
`client ping -c 10 -t 1 192.168.2.2`	Test time exceeded msg
`client traceroute 10.0.1.1`	Test traceroute



(a) ICMP

Tools for Disambiguating RFCs

Beyond Sage, there remains many challenges unaddressed.



Tools for Disambiguating RFCs

Jane Yen
University of Southern California
yeny@usc.edu

Ramesh Govindan
University of Southern California
ramesh@usc.edu

Barath Raghavan
University of Southern California
barathra@usc.edu

Abstract

For decades, drafting Internet protocols has taken significant amounts of human supervision due to the fundamental ambiguity of natural language. Given such ambiguity, it is also not surprising that protocol implementations have long exhibited bugs. This pain and overhead can be significantly reduced with the help of natural language processing (NLP).

We recently applied NLP to identify ambiguous or underspecified sentences in RFCs, and to generate protocol implementations automatically when the ambiguity is clarified. However this system is far from general or deployable. To further reduce the overhead and errors due to ambiguous sentences, and to improve the generality of this system, much work remains to be done. In this paper, we consider what it would take to produce a fully-general and useful system for easing the natural-language challenges in the RFC process.

CCS Concepts

• **Networks → Formal specifications**.

Keywords

natural language, protocol specifications

ACM Reference Format:
Jane Yen, Ramesh Govindan, and Barath Raghavan. 2021. Tools for Disambiguating RFCs. In *Applied Networking Research Workshop (ANRW '21), July 24–30, 2021, Virtual Event, USA.* ACM, New York, NY, USA, 7 pages. https://doi.org/10.1145/3472305.3472314

1 Introduction

It has long been the case that protocol behaviors are discussed and debated in the networking community for years before they are codified. The Internet Architecture Board (IAB) [9], Internet Engineering Task Force (IETF), and Internet Research Task Force (IRTF) [12] enable groups of independent networking professionals to draft Request for Comments (RFCs) to explain their networking ideas in English. Their aim is to publish their ideas globally and, typically, to codify those ideas as a networking standard. Draft RFCs are carefully evaluated and reviewed by members of working groups and editors; these individuals manually consider editorial and technical perspectives. Though the process by which RFCs are developed is effective, and can identify significant vague, incorrect, or partial descriptions and get authors to their drafts, the overall overhead of this human supervision is significant.

RFC editors follow a number of guidelines to review RFC drafts and provide feedback to the authors; authors typically aim to satisfy the guidelines as quickly as possible so that the back-and-forth review process can be minimized. The guidelines mostly address what sections are required to be present, what order the sections should be in, what minimal information should be stated clearly, and so forth. While these guidelines help drafts across time and space to maintain a uniform style, these guidelines are mostly writing advice and do not effectively help authors or reviewers to identify technical issues. To tackle technical concerns in any RFC draft requires participants' professional background knowledge.

RFC authors have many ways to describe technical details, including natural language, pseudocode, formal specifications, real code, state machine diagrams, and more. While pseudocode, formal specifications, and real code can provide precise execution steps, natural language is still preferable due to its flexibility and readability. However, natural language can also be fuzzy, which sometimes leads to a fundamental technical problem. For example, *"the type code changed to 0"* is a verbatim sentence in Internet Control Message Protocol (ICMP) RFC [23]. The noun phrase "type code" can confuse a reader as to whether it refers to as a type named "code" or a code named "type" or a specific protocol header field named "type code". If any reader interprets the noun phrase incorrectly, a packet may be generated incorrectly and get dropped by a receiver.

To strike the right balance when using natural language, we might ask the following question: **Can we systematically identify natural language ambiguity in network protocol specifications and make changes accordingly?** The answer to this question would help many stakeholders in the context of protocol specifications. For example, working groups and standards writers could leverage techniques to avoid or reduce back-and-forth

ANRW '21, July 24–30, 2021, Virtual Event, USA

ACM ISBN 978-1-4503-8618-8/21/07.
https://doi.org/10.1145/3472305.3472314

SAGE Limitations

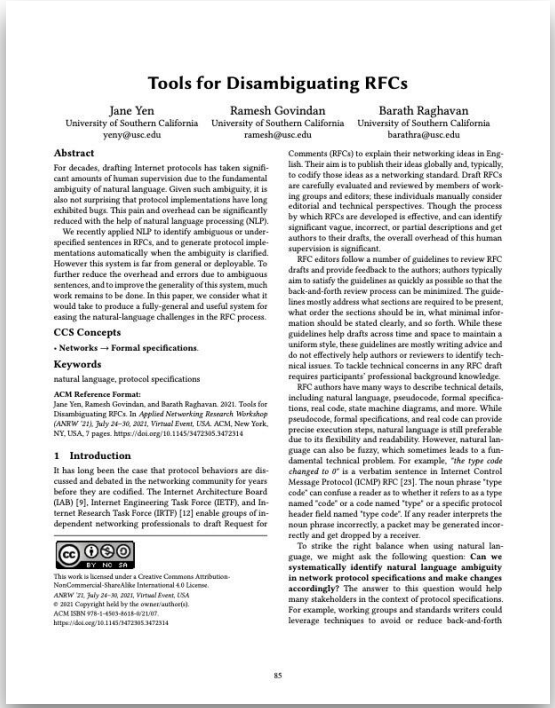
- Specification components

Name	Description
♦ **Packet Format**	Packet anatomy (i.e., field structure)
♦ **Field Descriptions**	Packet header field descriptions
♦ **Constraints**	Constraints on field values
♦ **Protocol Behaviors**	Reactions to external/internal events
System Architecture	Protocol implementation components
+ **State Management**	Session information and/or status
Comm. Patterns	Message sequences (e.g., handshakes)

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



Tools for Disambiguating RFCs

Jane Yen
University of Southern California
yeny@usc.edu

Ramesh Govindan
University of Southern California
ramesh@usc.edu

Barath Raghavan
University of Southern California
barathra@usc.edu

Abstract

For decades, drafting Internet protocols has taken significant amounts of human supervision due to the fundamental ambiguity of natural language. Given such ambiguity, it is also not surprising that protocol implementations have long exhibited bugs. This pain and overhead can be significantly reduced with the help of natural language processing (NLP).

We recently applied NLP to identify ambiguous or underspecified sentences in RFCs, and to generate protocol implementations automatically when the ambiguity is clarified. However this system is far from general or deployable. To further reduce the overhead and errors due to ambiguous sentences, and to improve the generality of this system, much work remains to be done. In this paper, we consider what it would take to produce a fully-general and useful system for easing the natural-language challenges in the RFC process.

CCS Concepts

• **Networks → Formal specifications**.

Keywords

natural language, protocol specifications

ACM Reference Format:
Jane Yen, Ramesh Govindan, and Barath Raghavan. 2021. Tools for Disambiguating RFCs. In *Applied Networking Research Workshop (ANRW '21), July 24–30, 2021, Virtual Event, USA.* ACM, New York, NY, USA, 7 pages. https://doi.org/10.1145/3472305.3472314

1 Introduction

It has long been the case that protocol behaviors are discussed and debated in the networking community for years before they are codified. The Internet Architecture Board (IAB) [9], Internet Engineering Task Force (IETF), and Internet Research Task Force (IRTF) [12] enable groups of independent networking professionals to draft Request for Comments (RFCs) to explain their networking ideas in English. Their aim is to publish their ideas globally and, typically, to codify those ideas as a networking standard. Draft RFCs are carefully evaluated and reviewed by members of working groups and editors; these individuals manually consider editorial and technical perspectives. Though the process by which RFCs are developed is effective, and can identify significant vague, incorrect, or partial descriptions and get authors to their drafts, the overall overhead of this human supervision is significant.

RFC editors follow a number of guidelines to review RFC drafts and provide feedback to the authors; authors typically aim to satisfy the guidelines as quickly as possible so that the back-and-forth review process can be minimized. The guidelines mostly address what sections are required to be present, what order the sections should be in, what minimal information should be stated clearly, and so forth. While these guidelines help drafts across time and space to maintain a uniform style, these guidelines are mostly writing advice and do not effectively help authors or reviewers to identify technical issues. To tackle technical concerns in any RFC draft requires participants' professional background knowledge.

RFC authors have many ways to describe technical details, including natural language, pseudocode, formal specifications, real code, state machine diagrams, and more. While pseudocode, formal specifications, and real code can provide precise execution steps, natural language is still preferable due to its flexibility and readability. However, natural language can also be fuzzy, which sometimes leads to a fundamental technical problem. For example, *"the type code changed to 0"* is a verbatim sentence in Internet Control Message Protocol (ICMP) RFC [23]. The noun phrase "type code" can confuse a reader as to whether it refers to as a type named "code" or a code named "type" or a specific protocol header field named "type code". If any reader interprets the noun phrase incorrectly, a packet may be generated incorrectly and get dropped by a receiver.

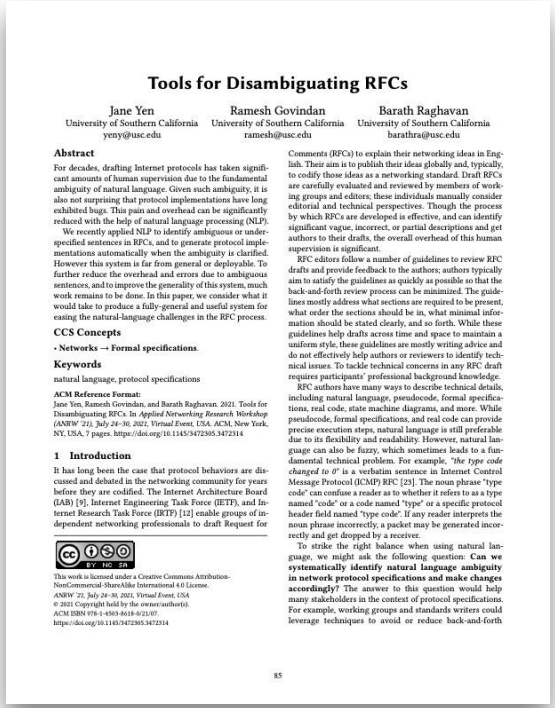
To strike the right balance when using natural language, we might ask the following question: **Can we systematically identify natural language ambiguity in network protocol specifications and make changes accordingly?** The answer to this question would help many stakeholders in the context of protocol specifications. For example, working groups and standards writers could leverage techniques to avoid or reduce back-and-forth

ANRW '21, July 24–30, 2021, Virtual Event, USA

ACM ISBN 978-1-4503-8618-8/21/07.
https://doi.org/10.1145/3472305.3472314

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



Tools for Disambiguating RFCs

Jane Yen
University of Southern California
yeny@usc.edu

Ramesh Govindan
University of Southern California
ramesh@usc.edu

Barath Raghavan
University of Southern California
barathra@usc.edu

Abstract

For decades, drafting Internet protocols has taken significant amounts of human supervision due to the fundamental ambiguity of natural language. Given such ambiguity, it is also not surprising that protocol implementations have long exhibited bugs. This pain and overhead can be significantly reduced with the help of natural language processing (NLP).

We recently applied NLP to identify ambiguous or underspecified sentences in RFCs, and to generate protocol implementations automatically when the ambiguity is clarified. However this system is far from general or deployable. To further reduce the overhead and errors due to ambiguous sentences, and to improve the generality of this system, much work remains to be done. In this paper, we consider what it would take to produce a fully-general and useful system for easing the natural-language challenges in the RFC process.

CCS Concepts

• **Networks → Formal specifications**.

Keywords

natural language, protocol specifications

ACM Reference Format:
Jane Yen, Ramesh Govindan, and Barath Raghavan. 2021. Tools for Disambiguating RFCs. In *Applied Networking Research Workshop (ANRW '21), July 24–30, 2021, Virtual Event, USA.* ACM, New York, NY, USA, 7 pages. https://doi.org/10.1145/3472305.3472314

1 Introduction

It has long been the case that protocol behaviors are discussed and debated in the networking community for years before they are codified. The Internet Architecture Board (IAB) [9], Internet Engineering Task Force (IETF), and Internet Research Task Force (IRTF) [12] enable groups of independent networking professionals to draft Request for Comments (RFCs) to explain their networking ideas in English. Their aim is to publish their ideas globally and, typically, to codify those ideas as a networking standard. Draft RFCs are carefully evaluated and reviewed by members of working groups and editors; these individuals manually consider editorial and technical perspectives. Though the process by which RFCs are developed is effective, and can identify significant vague, incorrect, or partial descriptions and get authors to their drafts, the overall overhead of this human supervision is significant.

RFC editors follow a number of guidelines to review RFC drafts and provide feedback to the authors; authors typically aim to satisfy the guidelines as quickly as possible so that the back-and-forth review process can be minimized. The guidelines mostly address what sections are required to be present, what order the sections should be in, what minimal information should be stated clearly, and so forth. While these guidelines help drafts across time and space to maintain a uniform style, these guidelines are mostly writing advice and do not effectively help authors or reviewers to identify technical issues. To tackle technical concerns in any RFC draft requires participants' professional background knowledge.

RFC authors have many ways to describe technical details, including natural language, pseudocode, formal specifications, real code, state machine diagrams, and more. While pseudocode, formal specifications, and real code can provide precise execution steps, natural language is still preferable due to its flexibility and readability. However, natural language can also be fuzzy, which sometimes leads to a fundamental technical problem. For example, *"the type code changed to 0"* is a verbatim sentence in Internet Control Message Protocol (ICMP) RFC [23]. The noun phrase "type code" can confuse a reader as to whether it refers to as a type named "code" or a code named "type" or a specific protocol header field named "type code". If any reader interprets the noun phrase incorrectly, a packet may be generated incorrectly and get dropped by a receiver.

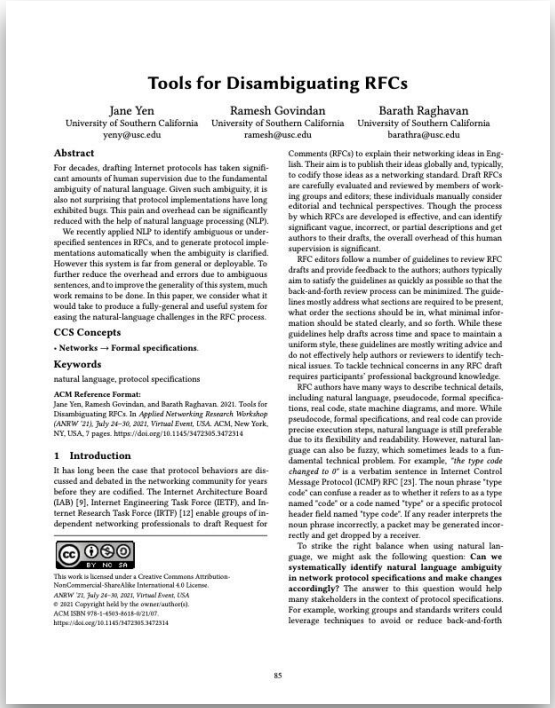
To strike the right balance when using natural language, we might ask the following question: **Can we systematically identify natural language ambiguity in network protocol specifications and make changes accordingly?** The answer to this question would help many stakeholders in the context of protocol specifications. For example, working groups and standards writers could leverage techniques to avoid or reduce back-and-forth

ANRW '21, July 24–30, 2021, Virtual Event, USA

ACM ISBN 978-1-4503-8618-8/21/07.
https://doi.org/10.1145/3472305.3472314

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



Tools for Disambiguating RFCs

Jane Yen
University of Southern California
yeny@usc.edu

Ramesh Govindan
University of Southern California
ramesh@usc.edu

Barath Raghavan
University of Southern California
barathra@usc.edu

Abstract

For decades, drafting Internet protocols has taken significant amounts of human supervision due to the fundamental ambiguity of natural language. Given such ambiguity, it is also not surprising that protocol implementations have long exhibited bugs. This pain and overhead can be significantly reduced with the help of natural language processing (NLP).

We recently applied NLP to identify ambiguous or underspecified sentences in RFCs, and to generate protocol implementations automatically when the ambiguity is clarified. However this system is far from general or deployable. To further reduce the overhead and errors due to ambiguous sentences, and to improve the generality of this system, much work remains to be done. In this paper, we consider what it would take to produce a fully-general and useful system for easing the natural-language challenges in the RFC process.

CCS Concepts

• **Networks → Formal specifications**.

Keywords

natural language, protocol specifications

ACM Reference Format:

Jane Yen, Ramesh Govindan, and Barath Raghavan. 2021. Tools for Disambiguating RFCs. In *Applied Networking Research Workshop (ANRW '21), July 24–30, 2021, Virtual Event, USA.* ACM, New York, NY, USA, 7 pages. https://doi.org/10.1145/3472305.3472314

1 Introduction

It has long been the case that protocol behaviors are discussed and debated in the networking community for years before they are codified. The Internet Architecture Board (IAB) [9], Internet Engineering Task Force (IETF), and Internet Research Task Force (IRTF) [12] enable groups of independent networking professionals to draft Request for Comments (RFCs) to explain their networking ideas in English. Their aim is to publish their ideas globally and, typically, to codify those ideas as a networking standard. Draft RFCs are carefully evaluated and reviewed by members of working groups and editors; these individuals manually consider editorial and technical perspectives. Though the process by which RFCs are developed is effective, and can identify significant vague, incorrect, or partial descriptions and get authors to their drafts, the overall overhead of this human supervision is significant.

RFC editors follow a number of guidelines to review RFC drafts and provide feedback to the authors; authors typically aim to satisfy the guidelines as quickly as possible so that the back-and-forth review process can be minimized. The guidelines mostly address what sections are required to be present, what order the sections should be in, what minimal information should be stated clearly, and so forth. While these guidelines help drafts across time and space to maintain a uniform style, these guidelines are mostly writing advice and do not effectively help authors or reviewers to identify technical issues. To tackle technical concerns in any RFC draft requires participants' professional background knowledge.

RFC authors have many ways to describe technical details, including natural language, pseudocode, formal specifications, real code, state machine diagrams, and more. While pseudocode, formal specifications, and real code can provide precise execution steps, natural language is still preferable due to its flexibility and readability. However, natural language can also be fuzzy, which sometimes leads to a fundamental technical problem. For example, *"the type code changed to 0"* is a verbatim sentence in Internet Control Message Protocol (ICMP) RFC [23]. The noun phrase "type code" can confuse a reader as to whether it refers to as a type named "code" or a code named "type" or a specific protocol header field named "type code". If any reader interprets the noun phrase incorrectly, a packet may be generated incorrectly and get dropped by a receiver.

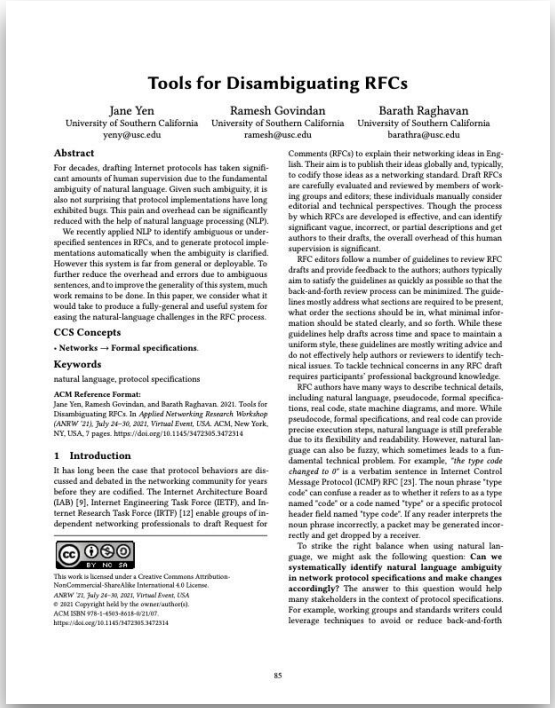
To strike the right balance when using natural language, we might ask the following question: **Can we systematically identify natural language ambiguity in network protocol specifications and make changes accordingly?** The answer to this question would help many stakeholders in the context of protocol specifications. For example, working groups and standards writers could leverage techniques to avoid or reduce back-and-forth

ANRW '21, July 24–30, 2021, Virtual Event, USA

ACM ISBN 978-1-4503-8618-8/21/07.
https://doi.org/10.1145/3472305.3472314

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



Tools for Disambiguating RFCs

Jane Yen
University of Southern California
yeny@usc.edu

Ramesh Govindan
University of Southern California
ramesh@usc.edu

Barath Raghavan
University of Southern California
barathra@usc.edu

Abstract

For decades, drafting Internet protocols has taken significant amounts of human supervision due to the fundamental ambiguity of natural language. Given such ambiguity, it is also not surprising that protocol implementations have long exhibited bugs. This pain and overhead can be significantly reduced with the help of natural language processing (NLP).

We recently applied NLP to identify ambiguous or underspecified sentences in RFCs, and to generate protocol implementations automatically when the ambiguity is clarified. However this system is far from general or deployable. To further reduce the overhead and errors due to ambiguous sentences, and to improve the generality of this system, much work remains to be done. In this paper, we consider what it would take to produce a fully-general and useful system for easing the natural-language challenges in the RFC process.

CCS Concepts

• **Networks → Formal specifications**.

Keywords

natural language, protocol specifications

ACM Reference Format:

Jane Yen, Ramesh Govindan, and Barath Raghavan. 2021. Tools for Disambiguating RFCs. In *Applied Networking Research Workshop (ANRW '21), July 24–30, 2021, Virtual Event, USA.* ACM, New York, NY, USA, 7 pages. https://doi.org/10.1145/3472305.3472314

1 Introduction

It has long been the case that protocol behaviors are discussed and debated in the networking community for years before they are codified. The Internet Architecture Board (IAB) [9], Internet Engineering Task Force (IETF), and Internet Research Task Force (IRTF) [12] enable groups of independent networking professionals to draft Request for Comments (RFCs) to explain their networking ideas in English. Their aim is to publish their ideas globally and, typically, to codify those ideas as a networking standard. Draft RFCs are carefully evaluated and reviewed by members of working groups and editors; these individuals manually consider editorial and technical perspectives. Though the process by which RFCs are developed is effective, and can identify significant vague, incorrect, or partial descriptions and get authors to their drafts, the overall overhead of this human supervision is significant.

RFC editors follow a number of guidelines to review RFC drafts and provide feedback to the authors; authors typically aim to satisfy the guidelines as quickly as possible so that the back-and-forth review process can be minimized. The guidelines mostly address what sections are required to be present, what order the sections should be in, what minimal information should be stated clearly, and so forth. While these guidelines help drafts across time and space to maintain a uniform style, these guidelines are mostly writing advice and do not effectively help authors or reviewers to identify technical issues. To tackle technical concerns in any RFC draft requires participants' professional background knowledge.

RFC authors have many ways to describe technical details, including natural language, pseudocode, formal specifications, real code, state machine diagrams, and more. While pseudocode, formal specifications, and real code can provide precise execution steps, natural language is still preferable due to its flexibility and readability. However, natural language can also be fuzzy, which sometimes leads to a fundamental technical problem. For example, *"the type code changed to 0"* is a verbatim sentence in Internet Control Message Protocol (ICMP) RFC [23]. The noun phrase "type code" can confuse a reader as to whether it refers to as a type named "code" or a code named "type" or a specific protocol header field named "type code". If any reader interprets the noun phrase incorrectly, a packet may be generated incorrectly and get dropped by a receiver.

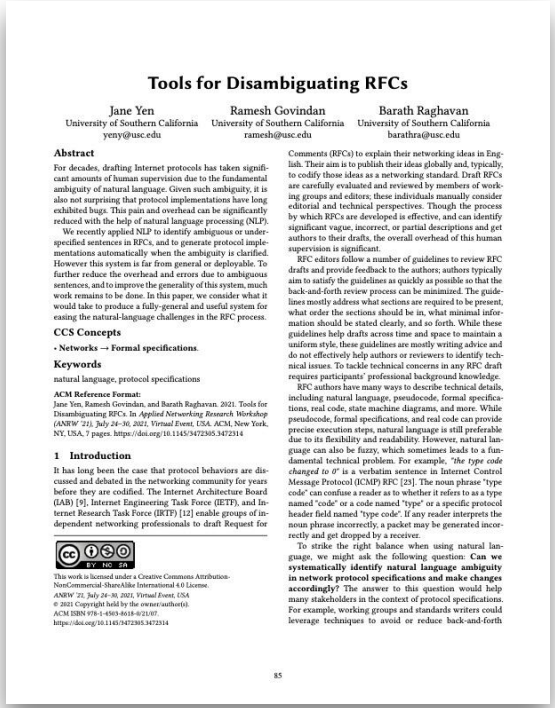
To strike the right balance when using natural language, we might ask the following question: **Can we systematically identify natural language ambiguity in network protocol specifications and make changes accordingly?** The answer to this question would help many stakeholders in the context of protocol specifications. For example, working groups and standards writers could leverage techniques to avoid or reduce back-and-forth

ANRW '21, July 24–30, 2021, Virtual Event, USA

ACM ISBN 978-1-4503-8618-8/21/07.
https://doi.org/10.1145/3472305.3472314

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



Tools for Disambiguating RFCs

Jane Yen
University of Southern California
yeny@usc.edu

Ramesh Govindan
University of Southern California
ramesh@usc.edu

Barath Raghavan
University of Southern California
barathra@usc.edu

Abstract

For decades, drafting Internet protocols has taken significant amounts of human supervision due to the fundamental ambiguity of natural language. Given such ambiguity, it is also not surprising that protocol implementations have long exhibited bugs. This pain and overhead can be significantly reduced with the help of natural language processing (NLP).

We recently applied NLP to identify ambiguous or underspecified sentences in RFCs, and to generate protocol implementations automatically when the ambiguity is clarified. However this system is far from general or deployable. To further reduce the overhead and errors due to ambiguous sentences, and to improve the generality of this system, much work remains to be done. In this paper, we consider what it would take to produce a fully-general and useful system for easing the natural-language challenges in the RFC process.

CCS Concepts

• **Networks → Formal specifications**.

Keywords

natural language, protocol specifications

ACM Reference Format:
Jane Yen, Ramesh Govindan, and Barath Raghavan. 2021. Tools for Disambiguating RFCs. In *Applied Networking Research Workshop (ANRW '21), July 24–30, 2021, Virtual Event, USA.* ACM, New York, NY, USA, 7 pages. https://doi.org/10.1145/3472305.3472314

1 Introduction

It has long been the case that protocol behaviors are discussed and debated in the networking community for years before they are codified. The Internet Architecture Board (IAB) [9], Internet Engineering Task Force (IETF), and Internet Research Task Force (IRTF) [12] enable groups of independent networking professionals to draft Request for Comments (RFCs) to explain their networking ideas in English. Their aim is to publish their ideas globally and, typically, to codify those ideas as a networking standard. Draft RFCs are carefully evaluated and reviewed by members of working groups and editors; these individuals manually consider editorial and technical perspectives. Though the process by which RFCs are developed is effective, and can identify significant vague, incorrect, or partial descriptions and get authors to their drafts, the overall overhead of this human supervision is significant.

RFC editors follow a number of guidelines to review RFC drafts and provide feedback to the authors; authors typically aim to satisfy the guidelines as quickly as possible so that the back-and-forth review process can be minimized. The guidelines mostly address what sections are required to be present, what order the sections should be in, what minimal information should be stated clearly, and so forth. While these guidelines help drafts across time and space to maintain a uniform style, these guidelines are mostly writing advice and do not effectively help authors or reviewers to identify technical issues. To tackle technical concerns in any RFC draft requires participants' professional background knowledge.

RFC authors have many ways to describe technical details, including natural language, pseudocode, formal specifications, real code, state machine diagrams, and more. While pseudocode, formal specifications, and real code can provide precise execution steps, natural language is still preferable due to its flexibility and readability. However, natural language can also be fuzzy, which sometimes leads to a fundamental technical problem. For example, *"the type code changed to 0"* is a verbatim sentence in Internet Control Message Protocol (ICMP) RFC [23]. The noun phrase "type code" can confuse a reader as to whether it refers to as a type named "code" or a code named "type" or a specific protocol header field named "type code". If any reader interprets the noun phrase incorrectly, a packet may be generated incorrectly and get dropped by a receiver.

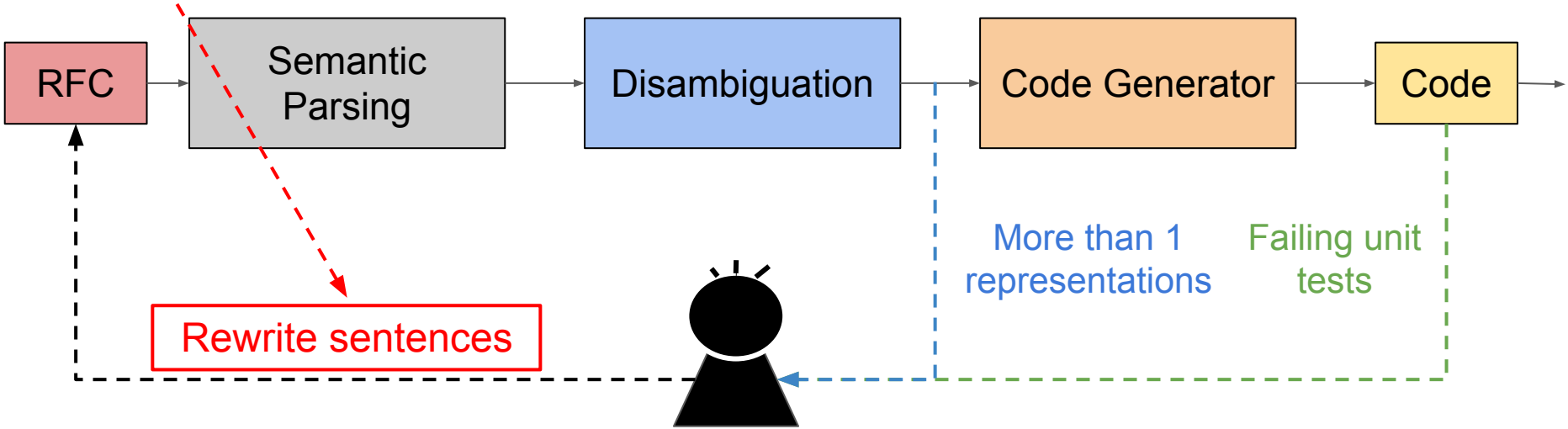
To strike the right balance when using natural language, we might ask the following question: **Can we systematically identify natural language ambiguity in network protocol specifications and make changes accordingly?** The answer to this question would help many stakeholders in the context of protocol specifications. For example, working groups and standards writers could leverage techniques to avoid or reduce back-and-forth

ANRW '21, July 24–30, 2021, Virtual Event, USA

ACM ISBN 978-1-4503-8618-8/21/07.
https://doi.org/10.1145/3472305.3472314

Current Work

Reduce Human Effort



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

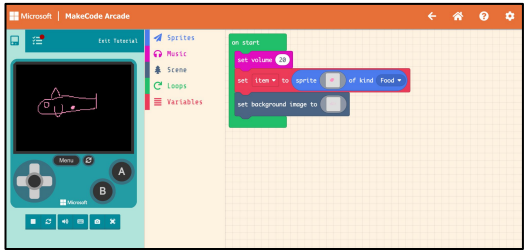
- What kind of protocols are we going to support?

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

Name	Description
♦ **Packet Format**	Packet anatomy (i.e., field structure)
♦ **Field Descriptions**	Packet header field descriptions
♦ **Constraints**	Constraints on field values
♦ **Protocol Behaviors**	Reactions to external/internal events
System Architecture	Protocol implementation components
+ **State Management**	Session information and/or status
Comm. Patterns	Message sequences (e.g., handshakes)

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

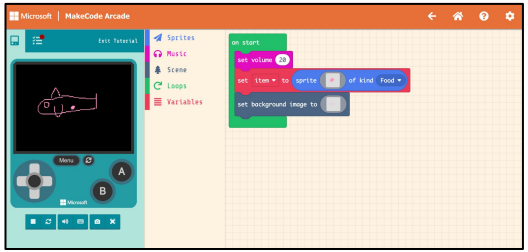
Our vision



User interface

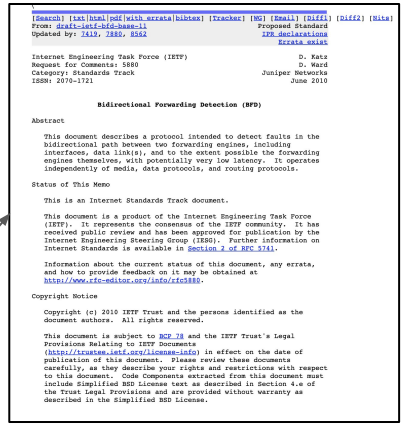
Essential protocol elements

Our vision



User interface

Essential protocol elements



[Search] [txt|html|pdf|with errata|bibtex] [Tracker] [WG] [Email] [Diff1] [Diff2] [Nits]
From: draft-ietf-bfd-base-11 Proposed Standard
Updated by: 7419, 7880, 8562 IPR declarations
Errata exist

Internet Engineering Task Force (IETF) D. Katz
Request for Comments: 5880 D. Ward
Category: Standards Track Juniper Networks
ISSN: 2070-1721 June 2010

Bidirectional Forwarding Detection (BFD)

Abstract

This document describes a protocol intended to detect faults in the bidirectional path between two forwarding engines, including interfaces, data link(s), and to the extent possible the forwarding engines themselves, with potentially very low latency. It operates independently of media, data protocols, and routing protocols.

Status of This Memo

This is an Internet Standards Track document.

This document is a product of the Internet Engineering Task Force (IETF). It represents the consensus of the IETF community. It has received public review and has been approved for publication by the Internet Engineering Steering Group (IESG). Further information on Internet Standards is available in Section 2 of RFC 5741.

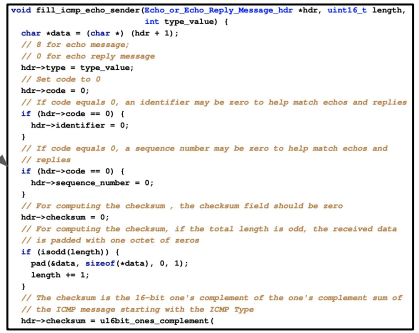
Information about the current status of this document, any errata, and how to provide feedback on it may be obtained at http://www.rfc-editor.org/info/rfc5880.

Copyright Notice

Copyright (c) 2010 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust's Legal Provisions Relating to IETF Documents (http://trustee.ietf.org/license-info) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must include Simplified BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.

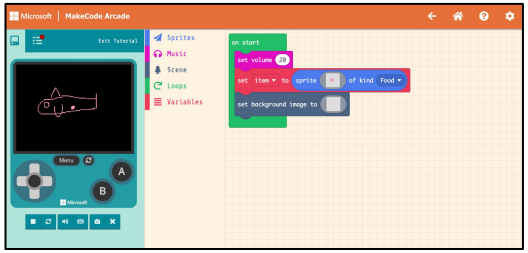
English RFC



```
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 = ul6bit_ones_complement(
```

Executable code

Our vision



User interface

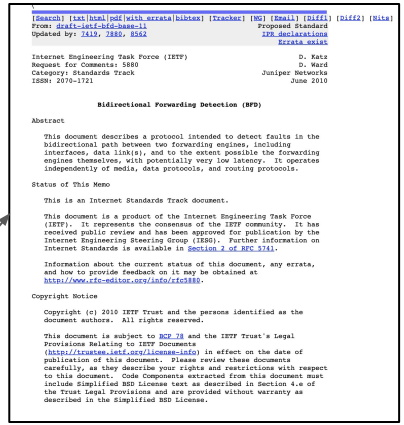
Timer

Last received packet

Output packet

Program states

Essential protocol elements



[Search] [txt|html|pdf|with errata|bibtex] [Tracker] [WG] [Email] [Diff1] [Diff2] [Nits]
From: draft-ietf-bfd-base-11 Proposed Standard
Updated by: 7419, 7880, 8562 IPR declarations
Errata exist

Internet Engineering Task Force (IETF) D. Katz
Request for Comments: 5880 D. Ward
Category: Standards Track Juniper Networks
ISSN: 2070-1721 June 2010

Bidirectional Forwarding Detection (BFD)

Abstract

This document describes a protocol intended to detect faults in the bidirectional path between two forwarding engines, including interfaces, data link(s), and to the extent possible the forwarding engines themselves, with potentially very low latency. It operates independently of media, data protocols, and routing protocols.

Status of This Memo

This is an Internet Standards Track document.

This document is a product of the Internet Engineering Task Force (IETF). It represents the consensus of the IETF community. It has received public review and has been approved for publication by the Internet Engineering Steering Group (IESG). Further information on Internet Standards is available in Section 2 of RFC 5741.

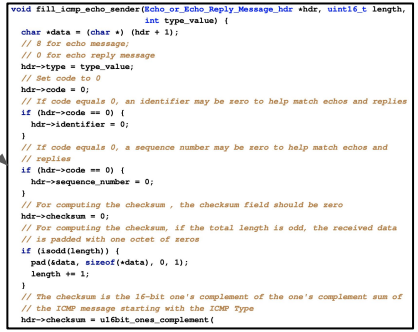
Information about the current status of this document, any errata, and how to provide feedback on it may be obtained at http://www.rfc-editor.org/info/rfc5880.

Copyright Notice

Copyright (c) 2010 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust's Legal Provisions Relating to IETF Documents (http://trustee.ietf.org/license-info) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must include Simplified BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.

English RFC



```
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 = ul6bit_ones_complement(
```

Executable code

Q & A

Yu-Chuan Yen

YENY@USC.EDU

SAGE

https://github.com/USC-NSL/sage.git