RFCScope: Detecting Logical Ambiguities in Internet Protocol Specifications

Mrigank Pawagi
Indian Institute of Science
Bengaluru, India
mrigankp@iisc.ac.in

Lize Shao
University of Virginia
Charlottesville, USA
zgr3et@virginia.edu

Hyeonmin Lee
University of Virginia
Charlottesville, USA
frv9vh@virginia.edu

Yixin Sun
University of Virginia
Charlottesville, USA
ys3kz@virginia.edu

Wenxi Wang
University of Virginia
Charlottesville, USA
wenxiw@virginia.edu

***Abstract*—Internet protocol specifications, published as Requests for Comments (RFCs) by the IETF organization, are essential to ensuring the interoperability, security, and reliability of the Internet. However, ambiguities in these specifications, particularly *logical ambiguities* such as inconsistencies and under-specifications, can lead to critical misinterpretations and implementation errors. Unfortunately, such ambiguities remain largely overlooked and challenging to detect with existing tools.**

In this paper, we present the first systematic study of verified technical errata from Standards Track RFCs over the past 11 years, identifying seven distinct subtypes of logical ambiguities. Building on these insights, we introduce RFCScope, the first scalable framework for detecting logical ambiguities in RFCs. RFCScope employs large language models (LLMs) through a modular pipeline that constructs targeted cross-document context, partitions specifications to preserve semantic integrity, applies bug-type-aware prompts for detection, and filters out false positives using structured reasoning validation.

RFCScope uncovers 31 new logical ambiguities spanning all seven subtypes across 14 recent RFCs. Eight of these have been confirmed by RFC authors, with three officially verified as technical errata. Our results demonstrate that RFCScope offers a practical solution for improving the clarity, consistency, and reliability of protocol standards through ambiguity detection.

I. Introduction

Internet protocols are the foundation of global digital communication, and their correctness is critical to ensuring interoperability, security, and reliability across diverse networked systems. The authoritative specifications for these protocols are published as Requests for Comments (RFCs) by the Internet Engineering Task Force (IETF) [1].

RFCs blend informal and formal elements, including natural language explanations, structured formal notations, diagrams, pseudocode, and tables, to clearly convey protocol specifications. Despite rigorous drafting, review, and iterative revision processes, the human-authored origin of RFCs inevitably introduces ambiguities that can undermine interoperability and implementation correctness across the Internet.

While prior work [2], [3] has explored detecting linguistic ambiguities in RFCs using manually written grammar rules, a critical gap remains: the identification of *logical ambiguities*, inconsistencies and under-specifications that span multiple sections, documents, or implicit assumptions. Logical ambiguities are especially concerning, as they directly impact protocol interpretation and implementation, yet remain largely under-explored and difficult to detect using existing approaches [2], [4], [5], [6], [7], [8], [9], [10], [11], [12].

To bridge this gap, we conduct the *first systematic study* of verified technical errata reports from Standards Track RFCs published over the past 11 years. Through manual analysis of 273 verified technical errata, we classify logical ambiguities into seven fine-grained subtypes, organized under two major categories (i.e., inconsistency and under-specification). Our analysis also reveals that these ambiguities frequently arise from multi-section dependencies, cross-document references, implicit domain assumptions, and feedback from practical implementations. This study lays a foundation for the design of automated tools to detect logical ambiguities in RFCs.

Recent advances in large language models (LLMs) offer promising new capabilities for automating specification analysis. However, applying LLMs to detect logical ambiguities in RFCs introduces several challenges: **Challenge 1**: RFCs are often lengthy and exceed the input limits of modern LLMs; **Challenge 2**: ambiguity detection requires context-aware reasoning across multiple interrelated documents; **Challenge 3**: LLMs often lack specialized domain knowledge about logical ambiguities in Internet protocols; and **Challenge 4**: LLMs are prone to hallucinating plausible but incorrect conclusions.

To address these challenges, we propose RFCScope, *the first framework* designed to detect logical ambiguities in RFCs. RFCScope introduces a scalable pipeline that combines four automated components: a **Context Constructor** that selectively extracts and synthesizes relevant context from both RFC and non-RFC references (addressing Challenges 1 and 2); a **Partitioner** that semantically segments the RFC and its context into manageable units suitable for LLMs (addressing Challenges 1 and 2); an **Analyzer** that detects ambiguities using bug-type-aware prompts (addressing Challenges 3 and 4); and an **Evaluator** that conservatively re-validates the Analyzer's reasoning to eliminate hallucinated or incorrect bug reports (addressing Challenge 4). A final manual inspection step further filters false positives to ensure correctness and practical relevance.

We evaluate RFCScope on 20 recent RFCs. The results show that RFCScope detects 31 new logical ambiguities across 14 RFCs, spanning all seven identified subtypes. So far, eight of these ambiguities have been confirmed by the corresponding RFC authors. Among them, three have been submitted to the IETF errata portal [13] as technical errata and officially verified.

Our contributions are summarized as follows:

Empirical Study: We present the *first* systematic study that characterizes logical ambiguities in verified RFC errata,

2643-1572/25/$31.00 ©2025 IEEE
DOI 10.1109/ASE63991.2025.00106

resulting in a fine-grained taxonomy that informs the design of automated detection methods.

Framework: We design and implement RFCScope, the *first* approach capable of systematically detecting logical ambiguities in RFCs.

Real-world Impact: RFCScope detected 31 new logical ambiguities in 14 recent RFCs. Eight of these ambiguities have been confirmed by RFC authors, and three are now officially verified errata [14], [15], [16].

Data Availability: To facilitate future research and tool development, we publicly release the full set of categorized errata reports from our study, the 31 ambiguity findings detected by RFCScope, and the prompt templates used in our analysis at: https://github.com/HIPREL-Group/RFCScope.

II. Background

A. Request for Comments (RFCs)

Request for Comments (RFCs) are technical documents that serve as the primary method for standardizing Internet protocols, published by Internet Engineering Task Force (IETF) [1]. Each RFC is assigned a unique number (e.g., RFC 9460 [17]) and remains immutable once published.

1) RFC Status: In the standardization process, protocol specifications typically begin as Internet-Drafts [18], which are subject to iterative review, revision, and community feedback. Once sufficiently mature, they may be published as RFCs on the *Standards Track*, the category used for protocols intended for widespread deployment and interoperability. To reflect a protocol's stability, implementation experience, and the level of community consensus, Standards Track RFCs progress through a series of maturity levels defined by the IETF [19], including:

- *Proposed Standard*: the first official stage; many protocols remain at this level while still being widely used in practice.
- *Draft Standard*: an intermediate stage that indicated greater maturity and implementation experience; this category is now deprecated and no longer used for new standards.
- *Internet Standard*: the final stage, indicating that the protocol is stable, interoperable, and widely deployed.

2) RFC Formats: RFC documents typically follow a structured format [20], organized into sections that begin with a metadata preamble and a table of contents. They often include standardized sections such as "IANA Considerations" [1] and "References" toward the end of the document.

To effectively convey protocol specifications, *RFC specifications blend informal and formal elements, including natural language, formal notations, tables, diagrams, pseudocode, and so on*, as shown in Figure 1. Most of the content is written in natural English, making the documents accessible to a wide audience. At the same time, RFCs frequently describe technical details such as message formats and protocol procedures using formal notations. Commonly used notations include Augmented Backus–Naur Form (ABNF) [21], Abstract Syntax Notation One (ASN.1) [22], Structure of Management Information (SMI) [23], Yet Another Next Generation (YANG) [24], and Concise Data Definition Language (CDDL) [25], [26]. In addition, RFCs may include pseudocode or actual code snippets to illustrate algorithmic procedures or implementation details. They also commonly use ASCII art diagrams and tables to visualize data layouts, protocol flows, or system architectures.

[1]IANA maintains registries for coordination between different organizations maintaining different parts of the Internet. For IETF, IANA allocates and maintains unique codes and numbering systems (parameters) used in the IETF technical standards.

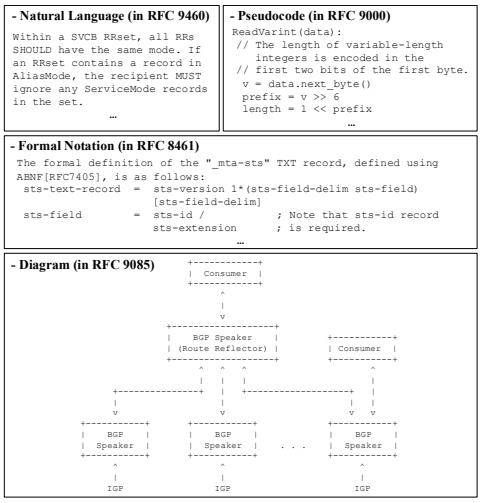

Fig. 1. Examples illustrate the diverse specification formats used across RFCs.

B. Errata Reports for RFCs

Errata reports are official records of errors identified in RFCs after publication, documenting technical, editorial, or procedural mistakes and offering corresponding corrections or clarifications [27]. Since RFCs are immutable once published, the errata mechanism provides a critical way to notify the community of known issues, preventing repeated reports and serving as a stopgap until an updated RFC is released.

Errata are submitted through an interface provided by the RFC Editor [28] and are classified into two types [27]:

- *Editorial*: errors in grammar, spelling, formatting, or similar.
- *Technical*: errors in the technical content of the RFC that affect its correct interpretation or implementation.

Each erratum is reviewed and assigned one of four statuses [27]:

- *Reported*: submitted but not yet reviewed.
- *Verified*: confirmed and corrected if necessary.
- *Rejected*: found to be incorrect or redundant.
- *Held for Document Update*: valid but not critical; to be considered in future revisions.

III. Related Work and Motivation

We describe existing work on investigating and formalizing protocol specifications. We then highlight the motivation of our work and how our approach differentiates from prior work.

A. Prior work on RFCs

RFC errata reports. A recent study [29] conducted a preliminary analysis on RFC errata reports to characterize the frequency of errata and reporting timelines. However, it did not examine the *content* of the errata reports. Our work takes the first step towards characterizing the types of ambiguity found in errata reports, which serves as the foundation for

TABLE I
CLASSIFICATION OF 273 VERIFIED TECHNICAL ERRATA REPORTS FROM THE PAST 11 YEARS VIA MANUAL INSPECTION

Main Category (Total Count)	Sub-Category	Count
Inconsistency (202)	(I-1) Direct inconsistency within or across specifications	119
	(I-2) Indirect inconsistency within or across specifications	70
	(I-3) Inconsistency with commonly accepted knowledge	13
Under-specification (37)	(U-1) Direct under-specification, due to undefined terms	7
	(U-2) Direct under-specification due to incomplete constraints (w.r.t. implementation feedback)	15
	(U-3) Indirect under-specification within or across specifications	10
	(U-4) Under-specification due to incorrect or missing references	5
Others (34)	Editorial errors	15
	IANA considerations	13
	Suggestions or proposals	6

developing automated methods to identify logical ambiguities in RFC documents.

Identify ambiguities in RFCs. The closest work to ours is Yen et al. [2], [3], which used a semantic-parsing approach to semi-automatically identify *linguistic* ambiguities in RFCs. However, this work does not consider *logical* ambiguities that are context-dependent, such as contradictory or under-specified statements, spanning across multiple sections, diagrams, or references. Furthermore, this work relies heavily on manually written grammar rules that are difficult to generalize and scale.

Automated formal modeling of RFCs. Pacheco et al. [4] used BERT to extract finite-state machines (FSMs) from RFC text, then converted these into Promela [37] for model checking. However, this work required manually annotated training data. To reduce the manual overhead, recent studies have explored LLMs, such as GPT-3.5 [38] and Code Llama [39], to translate RFC text into FSMs [40] or other formal specifications [41]. However, none of these works tackles the challenge of ambiguities in RFCs, which could result in missing or incorrect FSM transitions.

B. Prior work on detecting ambiguities in other NL documents

Besides RFCs, there have been studies that leverage LLMs to (semi-)automate the detection of ambiguities and contradictions in various protocol specification documents, such as IoT protocols (e.g., MQTT) [5], safety-critical systems [9], [10], and cellular network protocols (e.g., 3GPP) [11]. In addition to protocol specifications, there are studies that investigate the effectiveness of leveraging LLMs to detect ambiguities in other natural language (NL) documents [7], [12] or translate NL descriptions into linear temporal logic (LTL) statements [8].

C. Motivation for Our Work

Logical ambiguities in RFCs. Prior efforts to detect ambiguities in RFCs [2] focus primarily on *linguistic* ambiguities using *manually-crafted* grammar rules. However, a critical gap remains in identifying *logical ambiguities*, which can significantly impact the correct interpretation and implementation of protocols, posing risks to Internet interoperability.

Limitations in LLM-based approaches. Although LLMs have shown promise in detecting ambiguities in natural language texts [5], [7], [9], [10], [11], existing methods are typically tailored to single-document contexts and lack support for the dense cross-referencing common in RFCs. Approaches using BERT [4], rule-based parsing [2], or even GPT [40], [41] often suffer from limitations such as reliance on manual rules, hallucinations, or misinterpretations due to insufficient context. Logical ambiguities, which may span multiple sections and documents, present challenges that demand context-aware reasoning beyond what current automated techniques offer.

Our approach and contributions. We begin with the *first* systematic study of RFC errata *content* to identify and characterize logical ambiguities. These insights guide the design of RFCScope, our new LLM-based framework for detecting logical ambiguities in RFCs, addressing key limitations of prior work. To our knowledge, this is the *first* approach capable of systematically detecting logic-level ambiguities in RFCs at scale, an essential step toward improving protocol clarity and ensuring consistent implementation across the Internet.

We use the terminology "logically ambiguous bug", or simply "bug", interchangeably with "logical ambiguity" for the rest of the paper. Note that we use the term "bug" to broadly refer to any type of ambiguity in the protocol specification, which is different from software bugs.

IV. STUDY OF LOGICALLY AMBIGUOUS BUGS IN RFCS

To detect logical ambiguities in RFCs, it is essential to first understand the common types of bugs they contain. To this end, we conducted the *first systematic study* aimed at identifying prevalent types of logical ambiguity in RFCs published in recent years.

We began by collecting all Standards Track RFCs published in the past 11 years (from January 2014 to January 2025), resulting in 1,480 documents. We then gathered the corresponding errata reports by automatically scraping the errata pages using the RFC Editor's search interface [42], yielding 1,165 errata reports. Focusing specifically on technical issues that impact the correct interpretation or implementation of a specification, we *manually inspected* the errata reports that are classified as both Verified and Technical, totaling 273 entries.

Through this analysis, we identified three primary categories of bugs: 1) **Inconsistency**: contradictory information stated in the specification; 2) **Under-specification**: missing or incomplete information in the specification; and 3) Other: ambiguities that do not directly affect protocol semantics, including syntax or formatting issues, IANA compliance notes, or general editorial suggestions. The detailed breakdown of these categories is presented in Table I. Among the 273 bugs

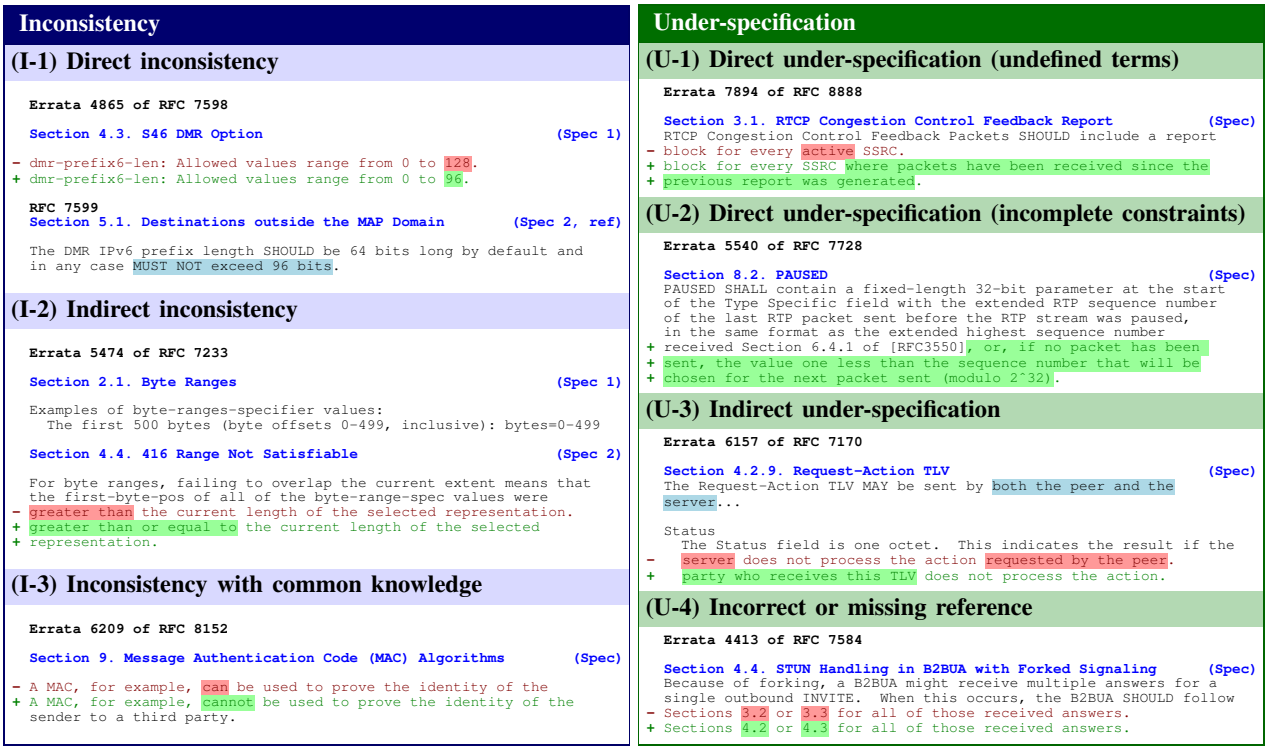

Fig. 2. Representative errata report examples for each bug sub-category identified in our study. I-1 RFC 7598 [30] Section 4.3 states that the `dmr-prefix6-len` parameter can range from 0 to 128. However, Section 5.1 of the referenced RFC 7599 limits this value to a maximum of 96, creating a direct contradiction. I-2 RFC 7233 [31] Section 4.4 states that a byte range does not overlap with a representation if its first byte position exceeds the representation's length. However, Section 2.1 implies that a representation of length N spans bytes 0 to $N-1$, so a range starting at byte N also does not overlap. For example, a 500-byte representation spans 0–499, making a range starting at 500 non-overlapping. I-3 RFC 8152 [32] Section 9 incorrectly claims that MACs can prove identity, despite the well-established fact in networking literature that MACs do not provide identity guarantees. U-1 RFC 8888 [33] uses the term `active` in Section 3.1 without providing a definition. U-2 RFC 7728 [34] Section 8.2 omits the case where a stream is paused before any packets are sent, hindering correct implementation. U-3 RFC 7170 [35] Section 4.2.9 does not specify the meaning of the `Status` field when the Request-Action TLV is sent by the peer, even though the section indicates that either party may send this TLV. U-4 RFC 7584 [36], Section 4.4 incorrectly refers to non-existent Sections 3.2 and 3.3.

we inspected, 202 fall under inconsistency, 37 under under-specification, and 34 under other.

We focus on discussing the first two categories, inconsistency and under-specification, as they pose the most significant risks to protocol clarity and correctness. In this study, we treat each section of an RFC as a distinct specification unit.

A. Inconsistency

We further classify inconsistency bugs into the following three sub-categories:

(I-1) Direct inconsistency within specifications: This sub-category includes bugs involving explicit contradictions. These may occur (1) within different parts of the same specification or (2) between specifications, where the conflicting content appears either elsewhere in the same RFC or in a referenced document (which may or may not be an RFC). For example, as shown in Figure 2 (I-1), RFC 7598 [30] Section 4.3 states that the `dmr-prefix6-len` parameter in the S46 DMR Option can take values from 0 to 128. However, the referenced Section 5.1 of RFC 7599 specifies that this value must not exceed 96, resulting in a direct contradiction (Errata 4865 [43]). We identified 119 bugs in this category.

(I-2) Indirect inconsistency within specifications: This category includes bugs involving contradictions that are not explicitly stated but can be inferred. As with direct inconsistencies, these may arise within the same specification or between specifications, where the conflicting content appears either elsewhere in the current RFC or in a referenced document. For instance, as illustrated in Figure 2 (I-2), RFC 7233 [31] Section 4.4 states that a byte range does not overlap with a representation if its first byte position is greater than the representation's length. However, Section 2.1 implies that a representation of length N spans bytes 0 to $N-1$, so a range starting at byte N also fails to overlap. For example, a 500-byte representation spans 0–499, and a range starting at 500 does not overlap (Errata 5474 [44]). 70 bugs are in this category.

(I-3) Inconsistency with common knowledge: These bugs arise when a specification contradicts widely accepted domain knowledge, independent of any referenced documents. For example, as shown in Figure 2 (I-3), RFC 8152 [32] Section 9 incorrectly states that MACs (Message Authentication Codes) can be used to prove identity, although it is well-established in the networking literature that MACs do not provide identity guarantees (Errata 6209 [45]). 13 bugs are in this subcategory.

B. Under-specification

We further classify the under-specification bugs into the following four subcategories:

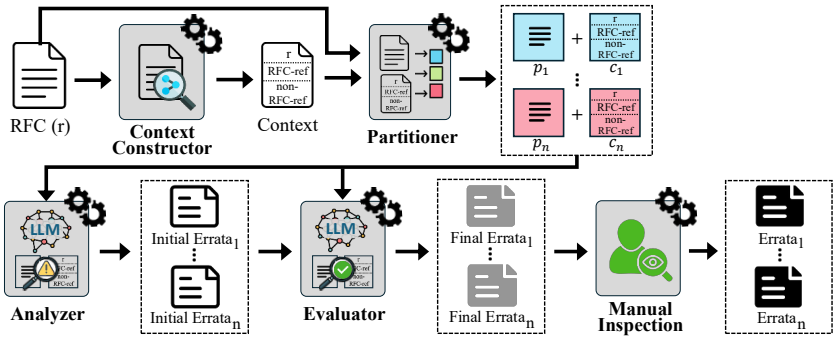

Fig. 3. Overview of RFCScope.

(U-1) Direct under-specifications due to undefined terms: This sub-category includes bugs involving explicitly missing information, specifically, undefined terms in the specification. The missing definitions cannot be found in the current RFC or any of its referenced documents. For example, as shown in Figure 2 (U-1), RFC 8888 [33] uses the term `active` in Section 3.1, but the document does not define it anywhere (Errata 7894 [46]). We identified 7 bugs in this category.

(U-2) Direct under-specification due to incomplete constraints (based on implementation feedback): This sub-category includes bugs where the specification omits crucial constraints needed for correct protocol implementation, such as unaddressed scenarios or missing cases. The missing information cannot be inferred from the current RFC or any referenced documents, and instead requires input from *implementation feedback* and discussions with the authors. For example, as demonstrated in Figure 2 (U-2), in RFC 7728 [34], Section 8.2 omits the situation where a stream is paused before any packets have been sent, which hinders correct implementation (Errata 5540 [47]). We identified 15 bugs in this category.

(U-3) Indirect under-specifications: This sub-category includes bugs involving implicitly missing information, such as incomplete constraints. These issues arise when: (1) a part of the specification lacks details provided elsewhere in the same specification, or (2) it omits information found in another specification, either within the current RFC or in a referenced document (which may or may not be an RFC). For example, as shown in Figure 2 (U-3), RFC 7170 [35] Section 4.2.9 does not specify the semantics of the `Status` field when the Request-Action TLV is sent by the peer. However, other parts of the section indicate that both the server and the peer (i.e., any party) may send this TLV (Errata 6157 [48]). We identified 10 bugs in this category.

(U-4) Incorrect/missing references: This sub-category includes two distinct cases: (1) references to non-existent sections, and (2) references to existing sections that do not contain the expected information. For instance, as illustrated in Figure 2 (U-4), RFC 7584 [36], Section 4.4 incorrectly refers to non-existent Sections 3.2 and 3.3 (Errata 4413 [49]). 5 bugs are in this category: three involved references that were not provided or were irrelevant, and two referred to non-existent sections.

C. Summary

By systematically identifying and categorizing these issues, we gain critical insight into the underlying causes of ambiguity. Overall, our study reveals that logical ambiguities in protocol specifications often stem from inconsistencies and under-specifications within the document itself, in relation to referenced documents, established domain knowledge, or practical implementation feedback. This understanding lays a foundation for our proposed approach to detecting logical ambiguities in protocol specifications.

V. RFCScope

In this section, we introduce RFCScope, an LLM-based approach designed to detect all the logically ambiguous bug types identified in our study.

A. Why Use LLMs?

As shown in the Background Section (Section II-A), RFCs pose a unique difficulty for automated analysis: they combine a variety of informal and formal elements, including natural language, formal notations, tables, diagrams, and pseudocode. Effectively detecting bugs in such documents requires the ability to understand and process this diverse range of content. Moreover, as discussed in the Study Section (Section IV), logically ambiguous bugs often depend on knowledge beyond the RFCs and referenced documents, including common network knowledge (bug type **I-3**), and practical implementation feedback (bug type **U-2**). Furthermore, many of these logic-level ambiguities, especially indirect ones (bug type **I-2** and **U-3**), demand reasoning capabilities that go beyond simple pattern matching or rule-based techniques.

These difficulties highlight the need for a detector that can handle heterogeneous content, leverage broad knowledge, and perform solid reasoning. LLMs are well-suited to meet these requirements, making them a natural choice as the engine of our approach to detecting logical ambiguities in RFCs.

B. Challenges of Using LLMs

However, applying LLMs to the task of detecting logically ambiguous bugs in RFCs introduces four challenges:

Challenge 1: Lengthy RFCs: RFCs are often lengthy, frequently exceeding the context window of even the most advanced LLMs. This limitation makes it challenging to input the entire document at once, resulting in incomplete analysis

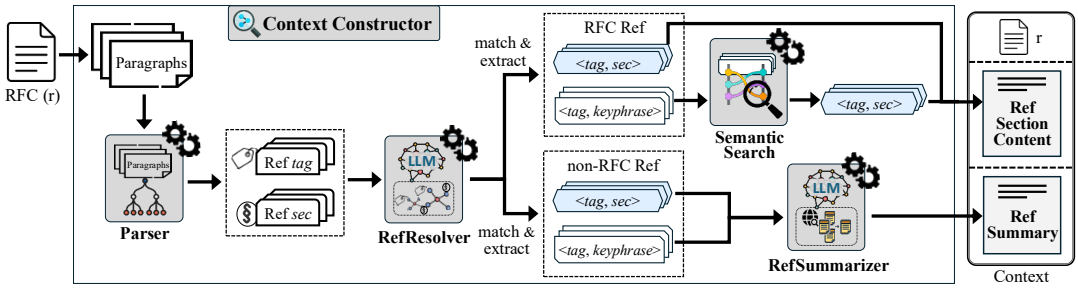

Fig. 4. Overview of Context Constructor.

and missed dependencies.

Challenge 2: Multi-document reasoning: Many bugs require reasoning across multiple referenced documents (bug type **I-1**, **I-2**, **U-2**, **U-3**, and **U-4**), placing additional strain on the model's ability to integrate context across inputs.

Challenge 3: Limited domain knowledge: While LLMs possess broad general knowledge, they often lack the specialized understanding needed to detect subtle, logically ambiguous bugs in network protocol specifications.

Challenge 4: Prone to hallucination: LLMs can generate plausible-sounding but incorrect statements. In the context of protocol analysis, such hallucinations can lead to false bug reports or misinterpretation of specification details.

C. Overview

Figure 3 illustrates the overall workflow of RFCScope. RFCScope is composed of four automated components: Context Constructor, Partitioner, Analyzer, and Evaluator.

To address Challenges 1 and 2 (i.e., the difficulty of processing lengthy RFCs and their references), RFCScope includes a Context Constructor to extract only the most relevant information from all referenced documents (including both RFCs and non-RFC documents), and a Partitioner to divide the RFC and its associated context into smaller, manageable segments. To tackle Challenge 3 (i.e., the LLM's limited domain-specific knowledge) and Challenge 4 (i.e., LLM hallucination), we guide the LLM-based Analyzer with well-defined bug types and examples identified in our study, helping the LLM focus on relevant ambiguities and reason more effectively within the domain of Internet protocol specifications. To further mitigate Challenge 4 (i.e., LLM hallucination), RFCScope includes an LLM-based Evaluator that verifies the reasoning steps behind each detected bug and filters out invalid ones.

Given an RFC document r, RFCScope proceeds as follows: 1) The Context Constructor extracts key contextual information from all referenced documents. 2) The Partitioner splits both the RFC r and its associated context into smaller, aligned partitions. 3) For each partition p and its corresponding context c, the LLM-based Analyzer identifies potential bugs and generates candidate bug reports. 4) These reports are then passed to the Evaluator, which checks the reasoning and filters out hallucinated or invalid cases. 5) Finally, the remaining reports are manually reviewed by human experts to produce the final bug reports.

We describe each component of RFCScope in detail in the following subsections.

```
9.6. Use of HTTPS RRs in Other Protocols

   All HTTP connections to named origins are eligible to use HTTPS RRs,
   even when HTTP is used as part of another protocol or without an
   explicit HTTP-related URI scheme (Section 4.2 of [HTTP]).  For
   example, clients that support HTTPS RRs and implement [WebSocket]
   using the altered opening handshake from [FETCH-WEBSOCKETS] SHOULD
   use HTTPS RRs for the requestURL.

15. References

15.1. Normative References

[HTTP] Fielding, R., Ed., Nottingham, M., Ed., and J. Reschke, Ed.,
       "HTTP Semantics", STD 97, RFC 9110, DOI 10.17487/RFC9110, June
       2022, <https://www.rfc-editor.org/info/rfc9110>.

[WebSocket] Fette, I. and A. Melnikov, "The WebSocket Protocol",
            RFC 6455, DOI 10.17487/RFC6455, December 2011,
            <https://www.rfc-editor.org/info/rfc6455>.

15.2. Informative References

[FETCH-WEBSOCKETS] WHATWG, "WebSockets Living Standard", September
                   2023, <https://websockets.spec.whatwg.org/>.
```

Fig. 5. Paragraph 1 of Section 9.6 from RFC 9460 [17]. This paragraph includes three reference tags: `HTTP`, `WebSocket`, and `FETCH-WEBSOCKETS`. The reference to `HTTP` explicitly includes a referenced section, `Section 4.2` (highlighted in red), while the other two do not specify sections. RFC references are highlighted in green, and non-RFC references in yellow. The phrase "opening handshake," highlighted in blue, is inferred as the keyphrase for the `WebSocket` and `FETCH-WEBSOCKETS` references by RefResolver.

D. Context Constructor

We define the context of an RFC document, or of a specific section within it, as the RFC document's content itself, along with key information from all referenced external sources, including both RFC and non-RFC documents. To address Challenges 1 and 2, our Context Constructor avoids including the full content of referenced documents, which is often too long for language models to process and typically unnecessary. Such content can overwhelm the model or introduce irrelevant distractions. Instead, we *selectively extract the key information explicitly or implicitly referred to by the input RFC*.

Figure 4 illustrates our overall workflow for extracting references. To enable fine-grained and accurate context construction, we first split the input RFC document into paragraphs. For each paragraph, we use IETF's `RFC2HTML` tool [50] to parse the content and extract all embedded *reference tags* and *referenced sections*. For example, as shown in Figure 5, a paragraph from RFC 9460 [17] contains three reference tags: `HTTP`, `WebSocket`, and `FETCH-WEBSOCKETS`, along with one explicitly referenced section: `Section 4.2`.

Given the extracted reference tags and referenced sections, we first use RefResolver (OpenAI's GPT-4o [51]) to associate reference tags with the corresponding referenced section numbers when they are explicitly provided (e.g., `4.2` for `HTTP` in Figure 5). For references without explicit section

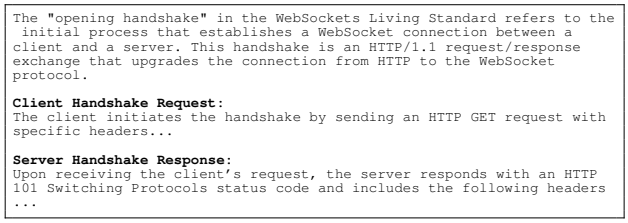

Fig. 6. Summary of the non-RFC reference to `FETCH-WEBSOCKETS` for the keyphrase "opening handshake," generated by RefSummarizer.

numbers (e.g., `WebSocket` and `FETCH-WEBSOCKETS`), we prompt GPT-4o to generate a *keyphrase* that captures the intended purpose of the reference. In Figure 5, the extracted keyphrase for both `WebSocket` and `FETCH-WEBSOCKETS` is "opening handshake." Note that we use GPT-4o for mapping reference tags to sections and extracting keyphrases because these tasks require only lightweight language understanding. GPT-4o performs them efficiently in a zero-shot setting while being faster and more cost-effective than reasoning models.

For RFC references, we extract only the *content of specific sections referred to*, based on our observation that RFCs almost always refer to particular sections, explicitly or implicitly, rather than the entire document. For RFC references with explicit section numbers, we directly retrieve the contents of the referenced sections. For RFC references with only keyphrases, we identify the most relevant sections using semantic search via LangChain [52], which leverages vector embeddings to match phrases with semantically similar document segments. For instance, for the `WebSocket` reference in Figure 5, semantic search retrieves Section 4.1 of RFC 6455 as the most relevant match for the keyphrase "opening handshake."

For non-RFC references, we extract summaries of the referred sections when available; otherwise, we summarize the entire document. In most cases, the input RFC refers only to specific concepts in non-RFC references rather than quoting exact content. We use RefSummarizer (OpenAI GPT-4o Search Preview [53]) to search for these documents online using their title and URL (when available). The model then generates a summary based on either the explicitly referenced section or, if no section is provided, the content relevant to the inferred keyphrase. For example, `FETCH-WEBSOCKETS` in Figure 5 is a non-RFC document without a specific section reference but is associated with the keyphrase "opening handshake." Using the keyphrase, the model produces a summary of the corresponding content, as shown in Figure 6. Note that we used GPT-4o Search Preview to summarize non-RFC references, as it was the only OpenAI model with web access at the time of submission. Since these references span diverse formats (webpages, PDFs, research papers), building custom scrapers was impractical. GPT-4o Search Preview reliably retrieved and summarized relevant content from a title or URL, leveraging its strength in "needle-in-a-haystack" tasks.

E. Partitioner

After the context is extracted, RFCScope employs the Partitioner component to split the RFC, along with its associated context, into smaller segments that fit within the context window of LLMs. This step directly addresses Challenge 1 and 2, which stems from the lengthy RFCs and their references.

Algorithm 1 Partitioning an RFC into smaller sections

Input: RFC r, context ctx
Output: Partitioned set P
Parameter: max section level d, max token length l
$P \leftarrow$ partition(r, ctx, d)
function PARTITION(p, ctx, d)
  if $d = 0$ **or** !p.hasSubsections **or** len$(p + ctx.get(p)) < l$
    return $\{p\}$
  else
    $P \leftarrow \emptyset$
    for each subsection p_{sub} in p
      $P \leftarrow P \cup$ partition$(p_{sub}, ctx, d-1)$
    end for
    return P
  end if
end function

RFCs are inherently hierarchical, organized into sections and nested subsections (e.g., Section 5, Section 5.1, Section 5.1.2). Rather than splitting the text arbitrarily or at fixed intervals, RFCScope leverages this structure to perform semantic-aware partitioning. Each partition aligns with the document's existing section boundaries, preserving the logical flow and intent of the specification.

To strike a balance between completeness and efficiency, the Partitioner avoids overly fine-grained splits (e.g., always breaking down to the lowest-level subsections), which would increase the number of model calls and reduce efficiency. Instead, it aims to preserve as much relevant information as possible within each partition while ensuring that the content remains within the model's token limit.

This process is guided by Algorithm 1, which recursively traverses the RFC's section hierarchy. For each section, it checks two conditions: 1) whether the section and its context fit within the LLM's token budget l, and 2) whether the section is within a user-defined maximum level d in the hierarchy. If a section satisfies these conditions—either because it's sufficiently small, has no further subsections, or the section level limit is reached—it is selected as a standalone partition. Otherwise, the algorithm recursively partitions its subsections, proceeding only as deep as necessary.

In summary, by selecting the largest possible self-contained sections with their relevant context that fit within the LLM input limits, the Partitioner achieves: 1) *logical coherence*, by aligning partitions with meaningful section boundaries; and 2) *processing efficiency*, by minimizing the number of model invocations. This hierarchical, token-aware partitioning strategy is a practical solution to the challenge of processing long, structured technical documents like RFCs, preserving their semantic integrity while enabling scalable and accurate LLM-based analysis.

F. Analyzer

After partitioning the RFC into manageable, context-rich segments, RFCScope uses its LLM-based Analyzer, powered by OpenAI's o3-mini [54], to detect logically ambiguous bugs. We selected o3-mini for its strong reasoning capabilities, which are critical for this task, requiring deep analysis such as explaining contradictions, validating under-specifications, and applying checklist-based criteria. Compared to full-sized reasoning models, o3-mini delivers strong chain-of-thought performance at lower cost. While GPT-4o is often used in LLM-as-a-Judge setups, recent work [55], [56] shows that o3-mini outperforms GPT-4o in reasoning-intensive evaluations. To address Challenge 3 (limited domain knowledge) and Challenge 4 (LLM hallucination), the Analyzer is guided by carefully designed prompts that incorporate domain-specific insights from our study of real-world RFC errata. This enables the model to generate initial errata reports.

For each partitioned section, the Analyzer is prompted twice: once to detect inconsistencies and once to detect under-specifications. These prompts incorporate all seven identified bug sub-categories (**I-1** to **I-3** and **U-1** to **U-4**) along with definitions and representative examples for each sub-category, enabling the model to reason explicitly about each potential logic-level ambiguity in RFCs.

A key feature of our design is that the LLM is explicitly instructed to perform chain-of-thought prompting [57] for every reported bug. This includes the concepts analyzed, the supporting evidence, and the logical steps used to reach a conclusion. This reasoning is not only valuable for interpretability; it is also essential for the Evaluator, which uses it to independently verify the model's output and filter out false positives.

By combining empirically identified bug categories, corresponding real-world examples, and structured reasoning instructions, RFCScope transforms a general-purpose LLM into a focused analysis engine for protocol specifications. This approach allows it to go beyond superficial keyword matching and instead emulate the kind of analytical rigor used in real-world errata reviews.

G. Evaluator

Once the initial errata reports are generated by the Analyzer, RFCScope invokes its LLM-based Evaluator, also powered by OpenAI's o3-mini [54] (chosen for the same reasons as in the Analyzer), to independently assess the validity of each report. This stage plays a critical role in addressing Challenge 4 (LLM hallucination) by acting as a conservative second-pass filter to improve the accuracy and credibility of the detected bugs.

A core design principle of RFCScope is that the Evaluator is explicitly instructed to validate the reasoning steps provided by the Analyzer. It must reconstruct its analysis from scratch, critically assess the justification, and determine whether the reported issue constitutes a real error in the RFC. This ensures that every reported bug is not only plausible, but also logically sound and independently validated.

The evaluation prompts enforce strict decision criteria grounded in the identified bug types from our study:

- Inconsistencies: The model is reminded that a valid report must present a clear and concrete contradiction that fits a recognized subcategory of inconsistency.
- Under-specifications: The model is guided to reject reports if: 1) the allegedly missing information appears later in the document, 2) the missing detail is irrelevant to the document's scope, or 3) the omission is intentional (e.g., left to implementer discretion or obvious from context). It must also verify that the issue fits a recognized subcategory of under-specification.

Only reports that pass this validation are retained as final potential errata. By coupling deep analysis with structured evaluation, RFCScope significantly reduces the risk of hallucinated bug reports. This layered design transforms general-purpose LLMs into a robust, self-checking system for discovering high-quality ambiguities in protocol specifications. Furthermore, the inclusion of detailed reasoning in each errata report simplifies subsequent manual inspection.

We make the prompt templates for both the Analyzer and Evaluator publicly available at https://github.com/HIPREL-Group/RFCScope.

H. Manual Inspection

We conduct a manual inspection to further eliminate false positives from the final set of potential errata. The manual step was conducted by two of the co-authors: a third-year undergraduate student and a recent graduate. On average, it took approximately 5 minutes to inspect each bug report generated by RFCScope.

This inspection is informed not only by the RFC itself and its references, but also by several supplementary sources: 1) **IETF Datatracker** [58]: provides access to all draft versions of the RFC and related working group discussions, allowing us to trace when a potential issue was introduced and whether it was previously addressed. This helped confirm bugs by identifying issues such as definitions removed in later drafts or inconsistent renaming across versions. 2) **IETF Mail Archive** [59]: archives mailing list discussions from various IETF working groups, offering deeper context and rationale behind specific edits or omissions during RFC development. We searched email threads to understand update rationales, verify whether a suspected bug was intentional or debated, and trace the history of unresolved issues. 3) **GitHub Repositories** [60]: some working groups manage RFC drafts on GitHub. The commit history, issues, and pull requests provide fine-grained insight into document changes and technical discussions. We used commit messages and blame annotations to date changes and find linked issues or pull requests for related discussions, and cross-referenced mail archives to clarify decisions. 4) **Supplemental References**: we consulted IANA registries, prior RFCs, and textbooks to clarify protocol parameters, terminology, and conventions.

RFCScope finally outputs the potential errata that remain valid after this manual inspection.

VI. Evaluation

In this section, we evaluate the effectiveness of RFCScope in detecting logical ambiguities for RFC specifications.

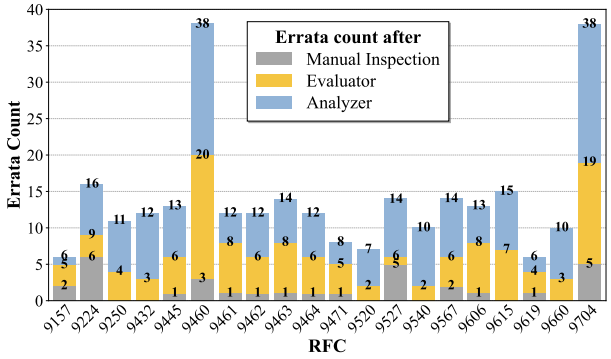

Fig. 7. Errata counts produced by individual components of RFCScope, Analyzer, Evaluator, and Manual Inspection, across the 20 target RFCs. Bar labels indicate cumulative values at each step of the pipeline.

A. Experimental Setup

1) Target RFCs: We evaluate RFCScope on the 20 most recent RFCs related to the Domain Name System (DNS), a core Internet protocol responsible for translating human-readable domain names into IP addresses. These RFCs, all classified as Proposed Standards and published by the IETF between November 2021 and the present, represent recent and actively maintained specifications in the networking domain. These include RFCs 9157 [61], 9224 [62], 9250 [63], 9432 [64], 9445 [65], 9460–9464 [17], [66], [67], [68], [69], 9471 [70], 9520 [71], 9527 [72], 9540 [73], 9567 [74], 9606 [75], 9615 [76], 9619 [77], 9660 [78], and 9704 [79].

2) Parameter Settings: For the Partitioner in RFCScope, we set the maximum section level $d = 3$ and the token budget $l = 175,000$ to ensure that each partition remains within the token limits of LLMs while preserving meaningful context. Importantly, limiting the section depth to $d = 3$ does not restrict RFCScope's ability to detect deeper ambiguities or affect the depth of semantic reasoning. We selected $d = 3$ based on empirical analysis of the evaluated RFCs, where this depth consistently produced semantically coherent partitions while remaining within the 200,000-token context window.

3) Bug Validation: To validate ambiguity findings generated by RFCScope, we perform manual investigation within our group and then contact the original RFC authors to seek feedback. If the authors encourage submitting an Errata Report for the finding, we then submit it to the IETF errata portal [13]. This direct engagement, encouraged by our IETF/DNS collaborator, provides richer feedback—for instance, one flagged finding was clarified as an intentional design choice under future discussion. However, the process is slow and labor-intensive, as authors may need to revisit old development history. To avoid overwhelming authors, particularly those involved in multiple RFCs, we are in the process of contacting them gradually.

4) Baselines: RFCScope is the first approach designed to detect logically ambiguous bugs in RFCs. However, as noted in Section III-B, four relevant approaches exist for ambiguity detection in natural language documents: Feng et al.[5], Mahbub et al.[9], Gärtner et al.[10], and Rahman et al.[11]. Among these, only Mahbub et al. [9] is both applicable and reproducible to RFCs, and we use it as our baseline.

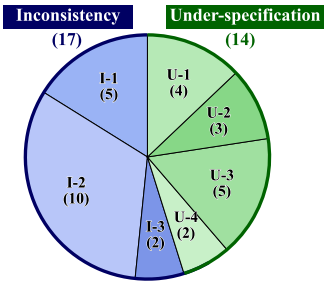

Fig. 8. Distribution of bugs detected by RFCScope across categories.

B. Research Questions

- **RQ1:** How many logical ambiguities can RFCScope detect?
- **RQ2:** What logical ambiguity types can RFCScope detect?
- **RQ3:** How do individual components contribute to the effectiveness of RFCScope?
- **RQ4:** How does RFCScope's performance compare to existing approaches?

C. RQ1: How many ambiguities can RFCScope detect?

As a result, RFCScope discovered a total of 31 new ambiguity findings across 14 out of the 20 targeted RFCs. None of the 31 reported ambiguities had been previously documented as errata. So far, eight ambiguities have been confirmed, categorized as follows: three fully confirmed and verified ambiguities; two (partially) confirmed, where clarifications would be beneficial but do not fall under the scope of an official Errata; and three that clarified the document's intended meaning but were not considered issues (e.g., intentional design choices). We have submitted the three fully confirmed ambiguities to the IETF errata portal [13] as technical errata, namely, Errata 8431 [14], Errata 8426 [15], and Errata 8590 [16]. All three Errata reports have been officially verified.

Figure 7 presents a detailed breakdown of how individual components, particularly the Analyzer and Evaluator, contribute to the detection across the 20 target RFCs. In total, the Analyzer produced 281 initial bug reports across all target RFCs, with the number of initial reports per RFC ranging from 6 (minimum, in RFC 9157) to 38 (maximum, in RFCs 9460 and 9704). From these 281 initial reports, the Evaluator filtered out 144 reports, yielding 137 potential errata across 20 RFCs. The number of potential errata per RFC ranged from 2 (minimum, in RFCs 9520 and 9540) to 20 (maximum, in RFC 9460). A final manual inspection step further removed 106 potential bug reports, resulting in a total of 31 final ambiguity findings.

In summary, RFCScope effectively identifies new logical ambiguities, demonstrating its practical value for improving the clarity of real-world RFC specifications.

D. RQ2: What types of ambiguities can RFCScope detect?

As a result, RFCScope successfully detects both inconsistency and under-specification ambiguities, *covering all seven identified subtypes* (**I-1** to **I-3**, **U-1** to **U-4**). Figure 8 shows the distribution of the 31 ambiguity findings detected across the 14 target RFCs: 17 are inconsistency bugs, including 5 direct (**I-1**), 10 indirect (**I-2**), and 2 against common knowledge (**I-3**);

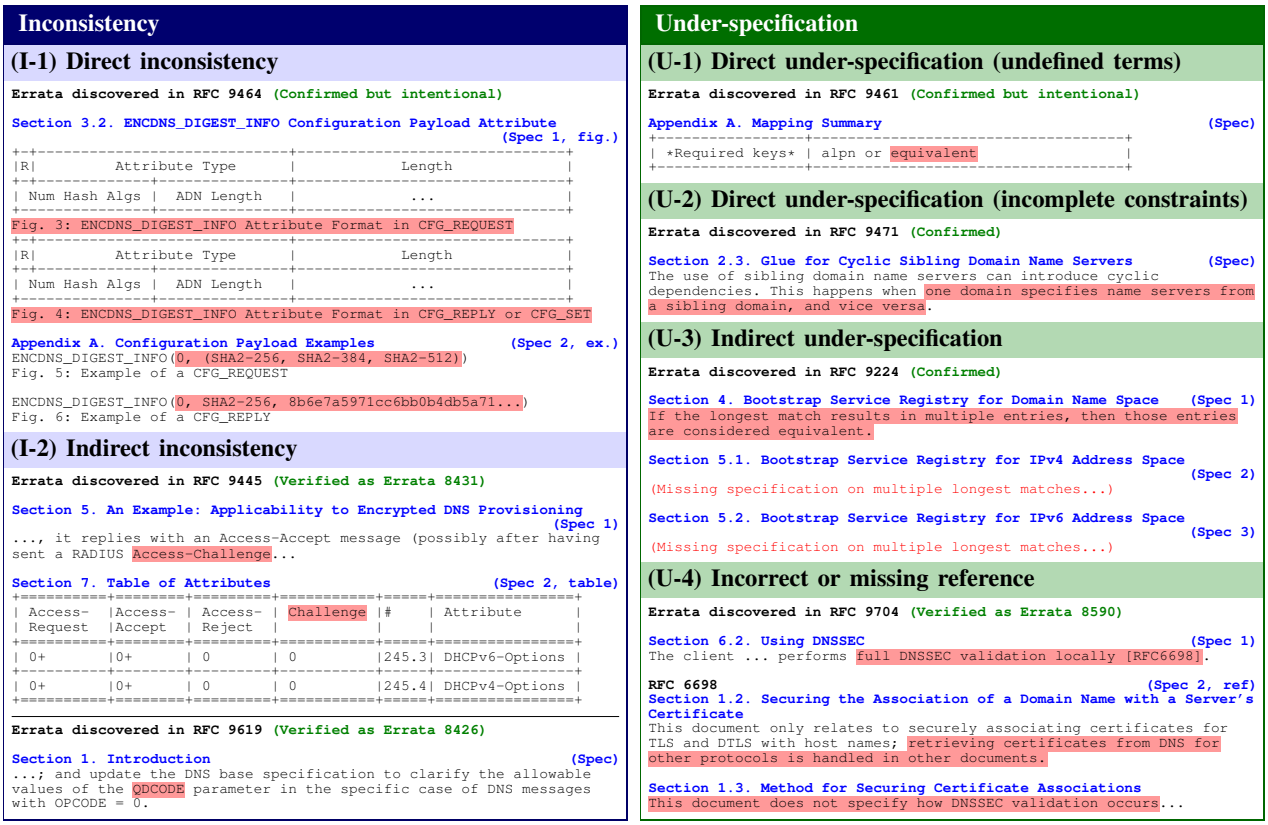

Fig. 9. Representative ambiguity example detected by RFCScope for each bug subtype. I-1 In RFC 9464 [69], Section 3.2 illustrates the fields of the ENCDNS_DIGEST_INFO attribute in CFG_REQUEST and CFG_REPLY. However, the examples in Appendix A do not show all fields, resulting in direct inconsistency. The authors confirmed that it is intentional to only include key fields in the examples. I-2 In RFC 9445 [65], Section 7 includes an attribute table that incorrectly lists "Challenge" instead of "Access-Challenge". This constitutes an indirect inconsistency, as it is clear from the rest of the document that "Access-Challenge" was the intended term. I-2 In RFC 9619 [77], Section 1 consistently refers to the field that indicates the number of questions in the Question section of a DNS message as "QDCOUNT", except in its last sentence where it instead refers to "QDCODE". It is clear from the rest of the document that this was not the intended term, resulting in an indirect inconsistency. U-1 Appendix A of RFC 9461 [66] specifies the required keys for the DNS SVCB mapping as "alpn or equivalent". Although the document mentions other keys that can indicate the value of "alpn" in its absence, it does not define any notion of "equivalence" to "alpn", resulting in an undefined term. The authors confirmed that this is an intentional placeholder for future protocol development. U-2 In RFC 9471 [70], Section 2.3 discusses cyclic sibling domain name servers but only addresses two-node cycles (i.e., between two domains). The lack of guidance on longer cycles results in an incomplete constraint. The authors confirmed that it would be beneficial to provide further explanation on cyclic dependencies. U-3 While Section 4 of RFC 9224 [62] addresses multiple longest matches for DNS, Section 5, covering IPv4 and IPv6, does not mention this scenario, suggesting an indirect under-specification. The authors confirmed that the multiple longest matches scenario serves as a safeguard mechanism and should apply to IPv4/IPv6 addresses (Section 5) as well. U-4 In RFC 9704 [79], Section 6.2 refers to RFC 6698 in the context of "full DNSSEC validation", but RFC 6698 specifies a different protocol and does not discuss DNSSEC validation. Therefore, this reference is incorrect.

and 14 are under-specification bugs, including 4 direct bugs due to undefined terms (**U-1**), 3 direct bugs due to incomplete constraints (**U-2**), 5 indirect (**U-3**), and 2 caused by incorrect or missing references (**U-4**).

Figure 9 presents a representative ambiguity example detected by RFCScope for each subtype, including three officially verified bugs and four confirmed bugs. Note that official errata reports include both the ambiguity and its proposed fix (as shown in Figure 2). Since RFCScope focuses solely on bug detection, it usually reports only the identified issue. For submitted errata, we incorporate fixes based on discussions with the RFC authors. From the shown bug examples, we can see that RFCScope achieves comprehensive coverage across a wide spectrum of logically ambiguous bug types, from subtle terminology gaps to complex cross-reference inconsistencies. *These results confirm RFCScope's capability to detect a diverse range of subtle, complex ambiguities across all identified bug categories, demonstrating its robustness and deep semantic understanding of protocol specifications.*

E. RQ3: How does each component contribute to RFCScope?

To evaluate the contribution of RFCScope's components, we conducted an ablation study with three variants: **A1**: removing domain-specific knowledge (i.e., identified bug-types and examples in our study as well as specialized instructions) from the Analyzer. **A2**: removing the Context Constructor. **A3**: removing the Partitioner. Figure 10 summarizes the results. Overall, each component is essential: removing any one substantially reduces the number of detected ambiguities. Among them, the Partitioner has the greatest impact, followed by the Context Constructor, and then domain-specific knowledge in the Analyzer. Furthermore, to evaluate whether the Evaluator

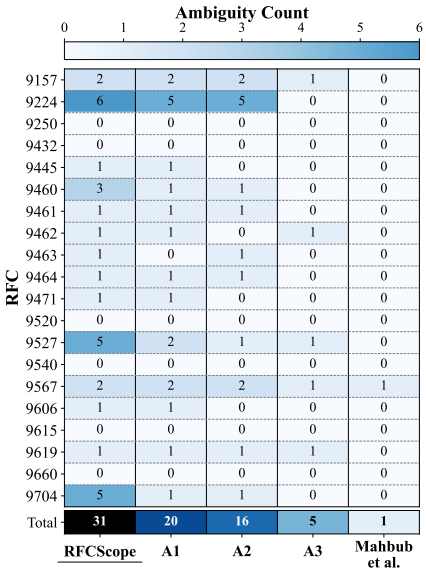

Fig. 10. Results against three variants and one existing approach: A1 excludes domain knowledge from the Analyzer; A2 excludes the Context Constructor; A3 excludes the Partitioner; and Mahbub et al. applies the approach from [9].

incorrectly discards potentially valid bugs, we conducted a focused analysis. We randomly selected 10 RFCs and manually reviewed all 71 initial bug reports that had been filtered out by the Evaluator. Among these, only one potentially valid bug was incorrectly discarded (as shown in Figure 11), demonstrating the Evaluator's high precision in preserving valid bugs while effectively eliminating false positives.

F. RQ4: How does RFCScope compare to existing approaches?

We applied the only applicable baseline, Mahbub et al.'s approach [9], to our 20 RFC subjects. As shown in Figure 10, it detected only one ambiguity, which is far fewer than RFCScope. Its limited performance likely results from (1) relying on very generic prompts without guidance on the types of inconsistencies and under-specifications in RFCs, and (2) not leveraging LLM reasoning through the commonly used chain-of-thought prompting technique [57].

VII. Discussion

Threats to validity. *1) False negatives.* Our work prioritizes precision but also considers recall. In our evaluation (Section VI-E), we manually reviewed 71 reports discarded by the Evaluator and found only one potentially valid bug across ten RFCs. This indicates a conservative, high-precision pipeline, though false negatives may remain. We plan to add a complementary Evaluator to mitigate this. *2) Completeness of bug types.* RFCScope follows a taxonomy derived from verified technical errata over the past 11 years, making ambiguities outside this scope less likely to be detected. While we are the first to formalize logic-level ambiguity categories, expanding to broader flaw types remains an important direction. *3) Context construction limitations.* RFCScope uses LLM-generated summaries for non-RFC references and semantic search for RFC content. This may introduce noise from

Errata identified by Analyzer but discarded by Evaluator in RFC 9445

Section 4. Passing RADIUS DHCP Options Attributes by DHCP Relay Agents to DHCP Servers (Spec 1)

Section 4.1. Context (Spec 2)

The RADIUS Attributes DHCP suboption [RFC4014] enables a DHCPv4 relay agent to pass identification and authorization attributes received during RADIUS authentication to a DHCPv4 server.

Fig. 11. The bug discovered by the Analyzer but discarded by the Evaluator. In RFC 9445, the title of Section 4 uses "DHCP", suggesting applicability to both DHCPv4 and DHCPv6. However, Section 4.1 only mentions "DHCPv4" without DHCPv6.

imprecise keyphrases or section mismatches, affecting detection quality. *4) Reproducibility.* As with many LLM-based systems, RFCScope depends on proprietary models that may evolve. Despite safeguards such as structured prompts, validation, and manual inspection, full reproducibility can be affected by API drift or model updates.

Future directions of RFC specification. RFCs are not written for a general audience, but rather a small set of domain experts and protocol implementers. This contributes to a considerable number of ambiguities we discovered, which would benefit from further specifications but could also be inferred implicitly by domain experts. However, with the rise of LLMs and the potential use of LLMs to interpret protocol specifications and generate implementations, the "users" of RFCs are silently shifting from (only) human experts to automated tools. Such trend calls the question on future requirements for RFC specifications, which likely need to be more precise and clearly-defined if they were to be interpreted by automated tools.

VIII. Conclusion

This paper introduces RFCScope, the first scalable framework for detecting logically ambiguous bugs in Internet protocol specifications. Grounded in the first systematic study of verified technical errata from Standards Track RFCs, RFCScope addresses a critical yet underexplored challenge: identifying subtle inconsistencies and under-specifications that undermine protocol clarity and interoperability. By leveraging LLMs through a carefully designed pipeline—spanning targeted context construction, semantic partitioning, bug-type-aware prompting, and structured reasoning validation—RFCScope demonstrates robust capabilities in uncovering logic-level ambiguities across a diverse set of recent RFCs. Its effectiveness is evidenced by the discovery of 31 previously undocumented ambiguity findings, several of which have already been confirmed or officially verified. RFCScope establishes a new standard for ambiguity detection in protocol specifications, and paves the way for more reliable, interpretable, and automatable Internet standards.

IX. Acknowledgment

We thank Shumon Huque for helping examine our findings and connect us with RFC authors. We thank all RFC authors for their thoughtful responses to our findings, including but not limited to Ben Schwartz, Dan Wing, Marc Blanchet, Mohamed Boucadair, Alan DeKok, Joe Abley, and Ray Bellis. This work is supported by National Science Foundation CNS-2154962 and CNS-2319421, and the Commonwealth Cyber Initiative.

References

[1] Internet Engineering Task Force (IETF). https://www.ietf.org/. (Accessed: September 30, 2025).

[2] Jane Yen, Tamás Lévai, Qinyuan Ye, Xiang Ren, Ramesh Govindan, and Barath Raghavan. Semi-automated protocol disambiguation and code generation. In *Proceedings of the ACM SIGCOMM 2021 Conference*. ACM, 2021.

[3] Jane Yen, Ramesh Govindan, and Barath Raghavan. Tools for disambiguating RFCs. In *Proceedings of the Applied Networking Research Workshop*. ACM, 2021.

[4] Maria Leonor Pacheco, Max von Hippel, Ben Weintraub, Dan Goldwasser, and Cristina Nita-Rotaru. Automated attack synthesis by extracting finite state machines from protocol specification documents. In *2022 IEEE Symposium on Security and Privacy*. IEEE, 2022.

[5] Yufei Feng, Yujie Zhang, and Yiming Chen. Detecting contradictions from IoT protocol specification documents based on neural generated knowledge graph. *Applied Sciences*, 2023.

[6] Alvaro Veizaga, Seung Yeob Shin, and Lionel C. Briand. Automated smell detection and recommendation in natural language requirements. *IEEE Transactions on Software Engineering*, 2024.

[7] Alessandro Fantechi, Stefania Gnesi, Lucia Passaro, and Laura Semini. Inconsistency detection in natural language requirements using ChatGPT: A preliminary evaluation. In *2023 IEEE International Requirements Engineering Conference*, 2023.

[8] Matthias Cosler, Christopher Hahn, Daniel Mendoza, Frederik Schmitt, and Caroline Trippel. nl2spec: Interactively translating unstructured natural language to temporal logics with large language models. In *Computer Aided Verification*, 2023.

[9] Taslim Mahbub, Dana Dghaym, Aadhith Shankarnarayanan, Taufiq Syed, Salsabeel Shapsough, and Imran Zualkernan. Can GPT-4 Aid in detecting ambiguities, inconsistencies, and incompleteness in requirements analysis? A comprehensive case study. *IEEE Access*, 2024.

[10] Thomas Gärtner and Daniel Göhlich. Automated requirement contradiction detection through formal logic and LLMs. *Automated Software Engineering*, 2024.

[11] Mirza Masfiqur Rahman, Imtiaz Karim, and Elisa Bertino. CellularLint: A systematic approach to identify inconsistent behavior in cellular network specifications. In *Proceedings of the 33rd USENIX Security Symposium*, 2024.

[12] Clement Guitton, Reto Gubelmann, Ghassen Karray, Simon Mayer, and Aurelia Tamò-Larrieux. Identifying open-texture in regulations using LLMs. *Artificial Intelligence and Law*, 2025.

[13] Internet Engineering Task Force. RFC editor errata search. https://www.rfc-editor.org/errata_search.php. (Accessed: September 30, 2025).

[14] RFC Editor. Errata report 8431 for RFC 9445: RADIUS extensions for DHCP-configured services. https://www.rfc-editor.org/errata/eid8431. (Accessed: September 30, 2025).

[15] RFC Editor. Errata report 8426 for RFC 9619: In the DNS, QDCOUNT is (usually) one. https://www.rfc-editor.org/errata/eid8426. (Accessed: September 30, 2025).

[16] RFC Editor. Errata report 8590 for RFC 9704: Establishing local DNS authority in validated split-horizon environments. https://www.rfc-editor.org/errata/eid8590. (Accessed: October 2, 2025).

[17] Benjamin M. Schwartz, Mike Bishop, and Erik Nygren. Service binding and parameter specification via the DNS (SVCB and HTTPS resource records). Internet Engineering Task Force, 2023. Request for Comments: 9460.

[18] Internet Engineering Task Force. Internet drafts. https://www.ietf.org/participate/ids. (Accessed: September 30, 2025).

[19] Christian Huitema, Jon Postel, and Steve Crocker. Not all RFCs are standards. Network Working Group, 1995. Request for Comments: 1796.

[20] Heather Flanagan and Sandy Ginoza. RFC style guide. Internet Architecture Board (IAB), 2014. Request for Comments: 7322.

[21] Dave Crocker and Paul Overell. Augmented BNF for syntax specifications: ABNF. Network Working Group, 2008. Request for Comments: 5234.

[22] ITU Telecommunication Standardization Sector. Introduction to ASN.1. https://www.itu.int/en/ITU-T/asn1/Pages/introduction.aspx. (Accessed: September 30, 2025).

[23] Jürgen Schönwälder, David T. Perkins, and Keith McCloghrie. Structure of management information version 2 (SMIv2). Network Working Group, 1990. Request for Comments: 2578.

[24] Martin Björklund. The YANG 1.1 data modeling language. Internet Engineering Task Force, 2016. Request for Comments: 7950.

[25] Henk Birkholz, Christoph Vigano, and Carsten Bormann. Concise data definition language (CDDL): A notational convention to express concise binary object representation (CBOR) and JSON data structures. Internet Engineering Task Force, 2019. Request for Comments: 8610.

[26] Internet Engineering Steering Group. Guidelines for the use of formal languages in IETF specifications, 2001.

[27] Alice Russo and Jean Mahoney. Current process for handling RFC errata reports. RFC Series Working Group, 2025. Internet-Draft: draft-rpc-errata-process-03.

[28] Internet Engineering Task Force. RFC editor: Report new errata. https://www.rfc-editor.org/errata.php. (Accessed: September 30, 2025).

[29] Stephen McQuistin, Mladen Karan, Prashant Khare, Colin Perkins, Matthew Purver, Patrick Healey, Ignacio Castro, and Gareth Tyson. Errare humanum est: What do RFC errata say about internet standards? In *2023 7th Network Traffic Measurement and Analysis Conference (TMA)*. IEEE, 2023.

[30] Tomek Mrugalski, Ole Trøan, Ian Farrer, Simon Perreault, Wojciech Dec, Congxiao Bao, Leaf Yeh, and Xiaohong Deng. DHCPv6 options for configuration of softwire address and port-mapped clients. Internet Engineering Task Force, 2015. Request for Comments: 7598.

[31] Roy T. Fielding, Yves Lafon, and Julian Reschke. Hypertext transfer protocol (HTTP/1.1): Range requests. Internet Engineering Task Force, 2014. Request for Comments: 7233.

[32] Jim Schaad. CBOR object signing and encryption (COSE). Internet Engineering Task Force, 2017. Request for Comments: 8152.

[33] Zaheduzzaman Sarker, Colin Perkins, Varun Singh, and Michael A. Ramalho. RTP control protocol (RTCP) feedback for congestion control. Internet Engineering Task Force, 2021. Request for Comments: 8888.

[34] Bo Burman, Azam Akram, Roni Even, and Magnus Westerlund. RTP stream pause and resume. Internet Engineering Task Force, 2016. Request for Comments: 7728.

[35] Hao Zhou, Nancy Cam-Winget, Joseph A. Salowey, and Steve Hanna. Tunnel extensible authentication protocol (TEAP) version 1. Internet Engineering Task Force, 2014. Request for Comments: 7170.

[36] Ram Ravindranath, Tirumaleswar Reddy.K, and Gonzalo Salgueiro. Session traversal utilities for NAT (STUN) message handling for SIP back-to-back user agents (B2BUAs). Internet Engineering Task Force, 2015. Request for Comments: 7584.

[37] Gerard J. Holzmann. The model checker SPIN. *IEEE Transactions on Software Engineering*, 1997.

[38] OpenAI Platform. GPT-3.5 Turbo. https://platform.openai.com/docs/models/gpt-3.5-turbo. (Accessed: September 30, 2025).

[39] Baptiste Roziere, Jonas Gehring, Fabian Gloeckle, Sten Sootla, Itai Gat, Xiaoqing Ellen Tan, Yossi Adi, Jingyu Liu, Romain Sauvestre, Tal Remez, et al. Code Llama: Open foundation models for code, 2023.

[40] Prakhar Sharma and Vinod Yegneswaran. PROSPER: Extracting Protocol Specifications Using Large Language Models. In *Proceedings of the 22nd ACM Workshop on Hot Topics in Networks*. ACM, 2023.

[41] Martin Duclos, Ivan A. Fernandez, Kaneesha Moore, Sudip Mittal, and Edward Zieglar. Utilizing Large Language Models to Translate RFC Protocol Specifications to CPSA Definitions, 2024.

[42] Internet Engineering Task Force. RFC editor search. https://www.rfc-editor.org/search/rfc_search.php. (Accessed: September 30, 2025).

[43] RFC Editor. Errata report 4865 for RFC 7598. https://www.rfc-editor.org/errata/eid4865. (Accessed: September 30, 2025).

[44] RFC Editor. Errata report 5474 for RFC 7233. https://www.rfc-editor.org/errata/eid5474. (Accessed: September 30, 2025).

[45] RFC Editor. Errata report 6209 for RFC 8152. https://www.rfc-editor.org/errata/eid6209. (Accessed: September 30, 2025).

[46] RFC Editor. Errata report 7894 for RFC 8888. https://www.rfc-editor.org/errata/eid7894. (Accessed: September 30, 2025).

[47] RFC Editor. Errata report 5540 for RFC 7728. https://www.rfc-editor.org/errata/eid5540. (Accessed: September 30, 2025).

[48] RFC Editor. Errata report 6157 for RFC 7170. https://www.rfc-editor.org/errata/eid6157. (Accessed: September 30, 2025).

[49] RFC Editor. Errata report 4413 for RFC 7584. https://www.rfc-editor.org/errata/eid4413. (Accessed: September 30, 2025).

[50] Internet Engineering Task Force. RFC2HTML. https://github.com/ietf-tools/rfc2html. (Accessed: September 30, 2025).

[51] OpenAI Platform. GPT-4o. https://platform.openai.com/docs/models/gpt-4o. (Accessed: September 30, 2025).

[52] LangChain. Build a semantic search engine. https://python.langchain.com/docs/tutorials/retrievers. (Accessed: September 30, 2025).
[53] OpenAI Platform. GPT-4o Search Preview. https://platform.openai.com/docs/models/gpt-4o-search-preview. (Accessed: September 30, 2025).
[54] OpenAI Platform. o3-mini. https://platform.openai.com/docs/models/o3-mini. (Accessed: September 30, 2025).
[55] Sijun Tan, Siyuan Zhuang, Kyle Montgomery, William Y. Tang, Alejandro Cuadron, Chenguang Wang, Raluca Ada Popa, and Ion Stoica. JudgeBench: A benchmark for evaluating LLM-based judges, 2025.
[56] Giulio Starace, Oliver Jaffe, Dane Sherburn, James Aung, Jun Shern Chan, Leon Maksin, Rachel Dias, Evan Mays, Benjamin Kinsella, Wyatt Thompson, Johannes Heidecke, Amelia Glaese, and Tejal Patwardhan. PaperBench: Evaluating AI's ability to replicate AI research, 2025.
[57] Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Fei Xia, Ed Chi, Quoc V Le, Denny Zhou, et al. Chain-of-thought prompting elicits reasoning in large language models. *Advances in neural information processing systems*, 2022.
[58] Internet Engineering Task Force. IETF datatracker. https://datatracker.ietf.org. (Accessed: September 30, 2025).
[59] Internet Engineering Task Force. IETF mail list archives. https://mailarchive.ietf.org/arch. (Accessed: September 30, 2025).
[60] Martin Thomson and Barbara Stark. Working group github usage guidance. Internet Engineering Task Force, 2020. Request for Comments: 8874.
[61] Paul E. Hoffman. Revised IANA considerations for DNSSEC. Internet Engineering Task Force, 2021. Request for Comments: 9157.
[62] Marc Blanchet. Finding the authoritative registration data access protocol (RDAP) service. Internet Engineering Task Force, 2022. Request for Comments: 9224.
[63] Christian Huitema, Sara Dickinson, and Allison Mankin. DNS over dedicated QUIC connections. Internet Engineering Task Force, 2022. Request for Comments: 9250.
[64] Peter van Dijk, Libor Peltan, Ondřej Surý, Willem Toorop, Kees Monshouwer, Peter Thomassen, and Aram Sargsyan. DNS catalog zones. Internet Engineering Task Force, 2023. Request for Comments: 9432.
[65] Mohamed Boucadair, Tirumaleswar Reddy.K, and Alan DeKok. RADIUS extensions for DHCP-configured services. Internet Engineering Task Force, 2023. Request for Comments: 9445.
[66] Benjamin M. Schwartz. Service binding mapping for DNS servers. Internet Engineering Task Force, 2023. Request for Comments: 9461.
[67] Tommy Pauly, Eric Kinnear, Christopher A. Wood, Patrick McManus, and Tommy Jensen. Discovery of designated resolvers. Internet Engineering Task Force, 2023. Request for Comments: 9461.
[68] Mohamed Boucadair, Tirumaleswar Reddy.K, Dan Wing, Neil Cook, and Tommy Jensen. DHCP and router advertisement options for the discovery of network-designated resolvers (DNR). Internet Engineering Task Force, 2023. Request for Comments: 9463.
[69] Mohamed Boucadair, Tirumaleswar Reddy.K, Dan Wing, and Valery Smyslov. Internet key exchange protocol version 2 (IKEv2) configuration for encrypted DNS. Internet Engineering Task Force, 2023. Request for Comments: 9464.
[70] Mark P. Andrews, Shumon Huque, Paul Wouters, and Duane Wessels. DNS glue requirements in referral responses. Internet Engineering Task Force, 2023. Request for Comments: 9471.
[71] Duane Wessels, William Carroll, and Matthew Thomas. Negative caching of DNS resolution failures. Internet Engineering Task Force, 2023. Request for Comments: 9520.
[72] Daniel Migault, Ralf Weber, and Tomek Mrugalski. DHCPv6 options for the homenet naming authority. Internet Engineering Task Force, 2024. Request for Comments: 9527.
[73] Tommy Pauly and Tirumaleswar Reddy.K. Discovery of oblivious services via service binding records. Internet Engineering Task Force, 2024. Request for Comments: 9540.
[74] Roy Arends and Matt Larson. DNS error reporting. Internet Engineering Task Force, 2024. Request for Comments: 9567.
[75] Tirumaleswar Reddy.K and Mohamed Boucadair. DNS resolver information. Internet Engineering Task Force, 2024. Request for Comments: 9606.
[76] Peter Thomassen and Nils Wisiol. Automatic DNSSEC bootstrapping using authenticated signals from the zone's operator. Internet Engineering Task Force, 2024. Request for Comments: 9615.
[77] Ray Bellis and Joe Abley. In the DNS, QDCOUNT is (usually) one. Internet Engineering Task Force, 2024. Request for Comments: 9619.
[78] Hugo Salgado, Mauricio Vergara Ereche, and Duane Wessels. The DNS zone version (ZONEVERSION) option. Internet Engineering Task Force, 2024. Request for Comments: 9660.
[79] Tirumaleswar Reddy.K, Dan Wing, Kevin Smith, and Benjamin M. Schwartz. Establishing local DNS authority in validated split-horizon environments. Internet Engineering Task Force, 2025. Request for Comments: 9704.