BBAE: Bit-to-Byte Alignment with Entropy Analysis for Binary Protocol Field Identification

by

Leijie Zhang
B.Sc., University of Victoria, 2023

A Thesis Submitted in Partial Fulfillment of the
Requirements for the Degree of

MASTER OF SCIENCE

in the Department of Computer Science

We acknowledge and respect the Lək̓ʷəŋən (Songhees and Xʷsepsəm/Esquimalt) Peoples on whose territory the university stands, and the Lək̓ʷəŋən and W̱SÁNEĆ Peoples whose historical relationships with the land continue to this day.

BBAE: Bit-to-Byte Alignment with Entropy Analysis for Binary Protocol Field Identification

by

Leijie Zhang
B.Sc., University of Victoria, 2023

Supervisory Committee

Dr. Kui Wu, Supervisor
(Department of Computer Science)

Dr. Jianping Pan, Departmental Member
(Department of Computer Science)

ABSTRACT

Protocol Reverse Engineering (PRE) is crucial for analyzing undocumented or proprietary network protocols, particularly in the fields of network security and the Internet of Things (IoT). To conserve network bandwidth, many protocols adopt a compact binary format that maximizes bit utilization. However, this compactness introduces significant challenges for PRE, because (1) the number of potential field boundaries grows exponentially, and (2) byte-oriented PRE tools become ineffective for these scenarios. To address these challenges, we propose Bit-to-Byte Alignment with Entropy (BBAE) analysis, an innovative approach designed to enhance boundary detection in bit-oriented protocols. BBAE leverages entropy analysis and bit-congruence calculations across multiple window sizes to identify field boundaries more effectively. In addition, it enables systematic verification of detected boundaries. We conducted extensive evaluations of BBAE's performance in identifying field boundaries of binary protocols and compared its effectiveness with existing tools, including byte-oriented semantic inference tools like BinaryInferno and bit-oriented tools such as Auto-ETLV. Our experimental results disclose that BBAE achieves outstanding performance in reverse engineering binary network protocols.

Contents

Supervisory Committee ii

Abstract iii

Table of Contents iv

List of Tables vi

List of Figures vii

Acknowledgements ix

1 Introduction 1

1.1 Background 1

1.2 Related Work 2

1.2.1 Byte-oriented Approach 3

1.2.2 Bit-oriented Approach 5

1.3 Contributions 6

2 Bit-to-Byte Alignment with Entropy Analysis (BBAE) 8

2.1 An Overview of BBAE 8

2.2 Step 1: Boundary Suggestion 9

2.2.1 Adjustable Window 11

2.2.2 Entropy and Similarity Analysis 12

2.2.3 Cross Verification 14

2.2.4 Step 2: Padding 20

2.2.5 Step 3: Byte-Aligned Analysis 20

2.2.6 Step 4: Ensembling 22

2.2.7 BBAE in Practice: An Illustrating Example 23

3 Evaluation Results **26**
3.1 How to Select a Suitable Threshold α? 27
3.2 Evaluation Results 27
3.3 Ablation Study of BBAE 30
3.3.1 Contribution of Adjustable Window and Entropy Analysis 32
3.3.2 Contribution of Padding and Byte-Aligned Analysis 32
3.3.3 Further Discussion 38

4 Graphical User Interface for BBAE **41**
4.1 Motivation and Design Goals 41
4.2 Prerequisites 42
4.3 Overview and Key Functionalities 42
4.4 Example Workflow 43

5 Conclusion and Future Work **45**
5.1 Conclusion 45
5.2 Future Work 47

Bibliography **48**

List of Tables

Table 3.1 Field Boundary Detection Results 31

List of Figures

Figure 2.1 The flowchart of BBAE. 10
Figure 2.2 Adjustable window size: Four different window sizes are used to analyze the information within a segment to infer potential field boundaries. 13
Figure 2.3 Entropy and similarity for TCP headers (larger windows). The blue line denotes similarity, and the red line denotes entropy. 17
Figure 2.4 Entropy and similarity for TCP headers (smaller windows). The blue line denotes similarity, and the red line denotes entropy. 18
Figure 2.5 Analysis of TCP header with BinaryInferno: The messages are in hexadecimal format, where every two characters constitute a byte. The boundaries returned from BinaryInferno are incorrect. For instance, the TCP header length field starts at the 96th bit and ends at the 100th bit, and BinaryInferno cannot detect this field because it assumes each segment is at least one byte long. 24

Figure 3.1 Precision-recall for TCP and MQTT protocols. 33
Figure 3.2 F1 score across protocols and tools. 34
Figure 3.3 Illustration of boundary detection for MQTT at various steps of BBAE, showcasing the progressive refinement of boundaries through Step 1 and Step 3. 35
Figure 3.4 Illustration of boundary detection for TCP at various steps of BBAE, showcasing the progressive refinement of boundaries through Step 1 and Step 3. 35
Figure 3.5 Entropy and similarity for MQTT headers (larger windows: 8 and 4). Blue line denotes similarity, red line denotes entropy. 36
Figure 3.6 Entropy and similarity for MQTT headers (smaller windows: 2 and 1). Blue line denotes similarity, red line denotes entropy. 37

Figure 4.1 Main user interface of the BBAE GUI. 42

Figure 4.2 Input panel of the BBAE GUI. Users can select a packet trace file by clicking the *Browse* button before running the analysis. 43
Figure 4.3 Input panel of the BBAE GUI. 44
Figure 4.4 Log panel of the BBAE GUI. 44

ACKNOWLEDGEMENTS

I would like to thank:

My friends, the lakes in Victoria where I went fishing, and even the weather, for being a source of comfort, humor, and resilience during the more difficult moments of this journey. Their presence helped me keep perspective and stay grounded.

My supervisor, Dr. Wu, for his invaluable mentoring, continuous support, encouragement, and patience throughout my research. His guidance has shaped both my academic work and my approach to learning.

All members of my academic community, who offered feedback, collaboration, and inspiration in countless ways that made this work possible.

Chapter 1

Introduction

1.1 Background

In today's interconnected world, network protocols are the foundation of network communication, specifying how devices exchange information across systems. However, the details of many protocols are unknown to the public since they are proprietary or undocumented [5]. This opacity presents a significant challenge, especially in critical areas like cybersecurity and system interoperability. Protocol Reverse Engineering (PRE) has become a critical method for addressing these challenges by deducing the hidden protocol specifications. Through the analysis of network traffic, PRE helps uncover unknown communication formats, ensuring compatibility between systems and strengthening security measures [2]. Many network services, such as attack detection, fuzz testing, and vulnerability analysis, heavily depend on PRE to manage risks and adapt to the changing network landscape [13, 14, 18]. As networks continue to grow in complexity, the demand for effective PRE techniques has intensified, making it a cornerstone in the modern network security domain.

PRE has made significant advances in recent years, leading to the development of various techniques. For example, BinaryInferno [3] leverages Shannon entropy to analyze field boundaries and infer semantic descriptions using multiple specialized detectors. Discoverer [4] identifies protocol idioms commonly found in application-level message formats by clustering and analyzing cross-field dependencies. More recently, advanced methods have integrated natural language processing (NLP) models to enhance protocol field boundary identification. For instance, TRIPRE employs transformers and keyword extraction techniques to distinguish internal protocol types [11]. Machine learning-based

approaches [19] [20] have further demonstrated their efficacy, as evidenced by recent research utilizing recurrent neural networks (RNNs) and hybrid architectures like U-Net and BiLSTM-CRF to analyze unknown traffic. These models excel at extracting both field boundaries and their semantic meanings [19] [20].

Despite their notable success, existing PRE tools face a significant challenge: a substantial portion of proprietary protocols are binary protocols, *meaning that their field boundaries can occur at arbitrary bit positions, not necessarily aligned to byte boundaries*. This intrinsic characteristic poses a major obstacle for most PRE tools, which typically rely on strong assumptions about byte alignment. For instance, BinaryInferno assumes that "fields are byte-aligned, with a minimum width of one byte, and that the data is not compressed or encrypted [3]." This assumption renders it ineffective for identifying sub-byte fields. Similarly, Discoverer [4] relies on tokens composed of consecutive bytes, inherently excluding bit-level granularity. Machine learning-based approaches also struggle with binary protocols due to their dependence on fixed-length tokens. These methods require consistent token sizes to maintain logical connections between input data and model interpretations, but this assumption is incompatible with the variable field positions and sizes of binary protocols. For example, models leveraging RNNs and hybrid architectures [19, 20], such as U-Net and BiLSTM-CRF, demonstrate strong performance on byte-aligned data but suffer significant accuracy loss with binary protocols. Similarly, TRIPRE's NLP-driven keyword extraction depends on predefined segment lengths, limiting its effectiveness for protocols with arbitrary field boundaries [11]. In summary, the design constraints of current PRE tools, particularly their reliance on fixed-length tokens and byte alignment, hinder their adaptability to the irregular and fine-grained structures of binary protocols. These limitations call for novel approaches that can accommodate the unique characteristics of binary protocol structures.

1.2 Related Work

The tasks involved in Protocol Reverse Engineering (PRE) can be categorized from various perspectives [7], depending on the specific objectives of the analysis and the characteristics of the protocols under consideration. Among these tasks, one of the most essential and widely studied problems is protocol field boundary detection, since accurate identification of field boundaries serves as the foundation for subsequent semantic inference and structural analysis. Without precise boundary detection, higher-level tasks such as field classification, semantic labeling, or protocol message reconstruction become

unreliable or even infeasible.

Focusing on the task of field boundary detection, existing approaches can be broadly divided into two main categories: byte-oriented methods and bit-oriented methods. Byte-oriented methods operate under the assumption that field boundaries are aligned to byte units. This assumption simplifies analysis and allows for the direct application of well-established tools and algorithms that work at the granularity of bytes. In contrast, bit-oriented methods remove this assumption and allow field boundaries to occur at arbitrary bit positions. This greater flexibility makes them more suitable for analyzing binary protocols that utilize compact field layouts and sub-byte structures. However, this flexibility also introduces substantial computational complexity, as the number of possible boundary positions increases exponentially when field boundaries are not restricted to byte alignment.

It is important to emphasize that, in this work, we focus on PRE methods that rely on network traces as their primary input. Network traces provide a passive and non-intrusive view of protocol behavior and capture how protocols operate in real-world environments. In contrast, approaches that depend on access to source code, binaries, or direct interaction with protocol implementations require privileged access or active manipulation of the target system, which is often unavailable or impractical for proprietary, legacy, or closed-source protocols. Such approaches therefore fall outside the scope of this categorization.

1.2.1 Byte-oriented Approach

Byte-oriented methods assume that protocol field boundaries align with byte units. Techniques such as TRIPRE, REInPro, DYNPre, NETPLIER, BinPRE, and BinaryInferno all fall in this category.

TRIPRE [11] uses a transformer-based framework for inferring protocol fields in industrial control protocols and utilizes a natural language processing model to cluster unknown industrial control protocol types. While it can generate half-byte-long tokens for its transformer model, it is unable to handle protocols with bit-length fields. REInPro [14] employs advanced techniques such as control field identification, clustering, and machine learning to efficiently infer the structure and semantics of industrial protocols from network traffic. DYNPre [12] provides dynamic inference by interacting with the server. Once the correct communication rules are identified, it becomes straightforward to reverse-engineer the formats and semantics of the messages sent by the server. However,

interacting with the server without prior knowledge of the communication specifications may be infeasible or at least very time-consuming. NETPLIER [17] leverages multiple sequence alignment (MSA) to identify keyword fields that determine message types and cluster protocol formats. BinPRE [9] incorporates an instruction-based semantic similarity analysis for format extraction and uses a cluster-and-refine paradigm to improve semantic inference accuracy further. Lastly, BinaryInferno employs multiple detectors to identify field boundaries and extract semantic information from protocols [3].

The above techniques have been demonstrated to be highly effective when the underlying assumption of byte alignment is satisfied, since the division of messages into fixed-size byte boundaries provides a natural structure for analysis. Under these circumstances, both statistical and learning-based methods can exploit the regularity of the data, yielding accurate and efficient boundary detection.

When this assumption does not hold, however, the performance of such approaches is significantly degraded. In protocols where fields may begin or end at arbitrary bit positions rather than at clean byte boundaries, existing byte-oriented methods often fail to capture the true structure of the message. In these cases, the lack of alignment results in tokens that do not correspond to semantically meaningful units, thereby introducing noise and reducing inference accuracy.

For example, machine learning-based approaches require that their input tokens contain sufficient structural information for the model to learn meaningful patterns. If the tokens are too short, such as individual bits, they fail to provide enough statistical evidence for the model to distinguish between different field types or boundaries. Consequently, both machine learning-based tools, such as TRIPRE and REInPro, require tokens that are at least half a byte in length to ensure that the training process captures recognizable patterns. Similarly, other representative tools, including NETPLIER and BinaryInferno, explicitly rely on dividing protocol messages into byte-long segments. These segments serve as the fundamental analysis units on which they compute entropy values or perform alignment across multiple message instances to identify consistent field boundaries.

Thus, while these methods demonstrate strong performance under the byte-alignment assumption, their effectiveness diminishes substantially when confronted with protocols that rely heavily on bit-level field boundaries, highlighting the need for alternative techniques that can operate at a finer granularity without sacrificing scalability or accuracy.

1.2.2 Bit-oriented Approach

Bit-oriented methods allow for field boundaries to occur at any arbitrary bit position, thereby removing the simplifying assumption of byte alignment and providing a more flexible framework for analyzing protocols with irregular or compact field layouts. This capability is particularly important in scenarios where protocol designers deliberately adopt non-standard field sizes to maximize efficiency or to obfuscate the protocol structure. By examining data at the granularity of individual bits, bit-oriented approaches can capture subtle variations in message formats that would otherwise be obscured if the analysis were constrained to byte-level segmentation.

To achieve this, such methods often rely on advanced computational techniques, including bitwise entropy calculations that measure the uncertainty within small bit sequences, as well as transition detection mechanisms that identify points in the message stream where statistical properties shift, indicating potential field boundaries. Additionally, some bit-oriented tools employ probabilistic models to account for variability in field lengths and to make boundary inference more robust against noise or limited training data.

Nevertheless, the increased flexibility of bit-level analysis comes at a cost. Because the number of possible boundary positions expands exponentially when each individual bit can represent a potential delimiter, bit-oriented methods typically require significantly more computational resources than their byte-aligned counterparts. This higher complexity can lead to scalability challenges, especially when analyzing large volumes of network traffic. Despite these challenges, bit-oriented methods remain a crucial class of techniques for accurately handling protocols with fine-grained field definitions, highlighting their importance in advancing the state of the art in PRE research.

PRE-Bin, a bit-oriented approach, employs clustering algorithms and a Bayesian decision model to delineate field boundaries and extract fields from binary frames [16]. Nevertheless, this method computationally prohibits analyzing a large traffic trace due to its complex Bayesian decision model. Auto-ETLV [8] defines a structure called the extended type-length-value (ETLV) structure, which can be found in many protocols. It analyzes the special properties of ETLV and uses these properties to reduce the search space of field boundaries significantly. Auto-ETLV is effective in identifying the length field and, accordingly, the (optional) type field and the value fields. Nevertheless, Auto-ETLV may not function properly if fields do not adhere to the ETLV format.

1.3 Contributions

So far, very few tools have been found to be effective for reverse engineering binary protocols. We aim to push the horizon of network trace-based binary PRE. This thesis introduces a novel approach, termed Bit-to-Byte Alignment with Entropy Analysis (BBAE), designed specifically for the challenges of binary protocol reverse engineering. Unlike existing methods that rely solely on either byte-oriented or bit-oriented assumptions, BBAE is constructed to combine the advantages of both paradigms into a single, cohesive framework. By doing so, it can address the limitations inherent in each individual category of methods and provide a more comprehensive solution for field boundary detection.

At its core, BBAE develops a new bit-oriented technique that systematically scans protocol messages to identify potential field boundaries at the bit level. These candidate boundaries are then transformed through a reversible padding mechanism, which extends sub-byte fields to align with full byte boundaries. This padding step is critical, as it not only preserves the original semantics of the protocol but also ensures compatibility with established byte-oriented analysis tools, which have traditionally demonstrated strong performance when applied to byte-aligned data. After the paddings, we leverage the rich solutions for analyzing byte-oriented protocols to infer protocol fields. Finally, we remove the paddings to recover the original field structure of the binary protocol.

Unlike existing tools, which rely on rigid assumptions about alignment or protocol format, our approach uses a window of adjustable size and calculates entropy and bit-congruence. In this way, we ensure comprehensive bit-level coverage and accurate boundary detection while maintaining compatibility with existing byte-aligned solutions. The novelty of our approach lies in its ability to seamlessly integrate diverse byte-aligned PRE tools for reverse engineering bit-oriented binary protocols. This offers a powerful and scalable solution for tackling the longstanding issues in reverse engineering binary protocols that have arbitrary bit-level field boundaries.

The main contributions of this thesis are as follows:

- **Novel boundary padding technique**: We propose a method to dynamically pad high-potential boundary bits in binary protocols, enabling compatibility with existing byte-oriented PRE tools while maintaining the original protocol structure and ensuring reversibility.

- **Integrated verification approach**: We develop an adjustable window technique,

combined with entropy and bit-congruence calculations and cross-verification, to detect and verify field boundaries in binary protocols.

- **Leveraging strength of both worlds**: By bridging bit-oriented and byte-oriented approaches, our method leverages the strengths of existing PRE solutions, such as BinaryInferno and Auto-ETLV, to analyze binary protocols more effectively.
- **Comprehensive framework**: We design a pipeline algorithm that integrates diverse PRE methods to ensure robust field boundary detection and scalable performance for binary PRE. We validate the effectiveness of our solution with two representative binary protocols, TCP and MQTT.

Chapter 2

Bit-to-Byte Alignment with Entropy Analysis (BBAE)

2.1 An Overview of BBAE

BBAE presents a comprehensive approach to Protocol Reverse Engineering (PRE) that is explicitly tailored for the challenges of binary protocols with bit-level field boundaries. Traditional PRE tools frequently rely on the assumption that fields are byte-aligned, an assumption that simplifies analysis but also severely limits their applicability. When field boundaries occur at arbitrary bit positions, these conventional methods become ineffective, often failing to detect important sub-byte fields or misclassifying protocol structures. BBAE addresses this fundamental limitation by integrating multiple complementary techniques: entropy analysis, bit-congruence measurements, existing byte-oriented tools, and a reversible padding mechanism into a unified framework that can transform and align bit-level information for effective analysis at the byte level. Another important design choice of BBAE is that it operates purely on network traces as its input. This choice is not incidental, but rather stems from the nature of the signals exploited by the proposed method. BBAE relies on statistical properties such as entropy variation and bit-level similarity across multiple message instances, which can only be observed by analyzing protocol behavior as it appears on the wire. These properties emerge from real traffic variability and cannot be reliably inferred from a single execution path or isolated message instance. By using network traces, BBAE is able to capture natural variations caused by different configurations, message types, and runtime conditions, all of which are essential for robust boundary detection.

The high-level architecture of the BBAE framework is summarized in Fig. 2.1, which consists of four major steps designed to progressively refine and validate field boundary detection:

1. **Boundary Suggestion:** Raw data, initially captured in hexadecimal format, is converted into a bit sequence to expose fine-grained structural details. Entropy analysis with adjustable windows is then applied, allowing the framework to infer potential boundary positions at multiple granularities. This step identifies candidate boundaries by capturing changes in randomness and similarity across segments. The top three boxes in Fig. 2.1 illustrate this process in detail.

2. **Padding:** The boundaries obtained from Step 1 often include false positives due to the sensitivity of entropy and similarity measures. To address this, we introduce a padding algorithm that extends sub-byte fields into byte-aligned segments. This ensures compatibility with byte-oriented tools while preserving the semantics of the original protocol structure. The fourth box in Fig. 2.1 represents this step.

3. **Byte-Aligned Analysis:** Once the fields have been padded, the data is converted back into hexadecimal format and processed by state-of-the-art byte-oriented analysis tools, such as BinaryInferno. This step provides an additional layer of validation and can reveal boundaries that were difficult to detect using entropy and similarity analysis alone. After analysis, the reversible padding is removed so that the results can be mapped back to the true bit-level structure. The fifth box in Fig. 2.1 shows this step.

4. **Ensembling:** In the final step, results from entropy analysis, bit-congruence evaluation, and byte-aligned validation are merged into a consolidated output. Cross-verification is employed to reduce uncertainty and eliminate spurious detections, resulting in a robust set of final boundaries. The bottom two boxes in Fig. 2.1 illustrate this stage.

2.2 Step 1: Boundary Suggestion

This step is responsible for identifying potential field boundaries within the binary protocol message, forming the foundation for subsequent analysis. The process begins with the application of an adjustable window mechanism, which systematically divides

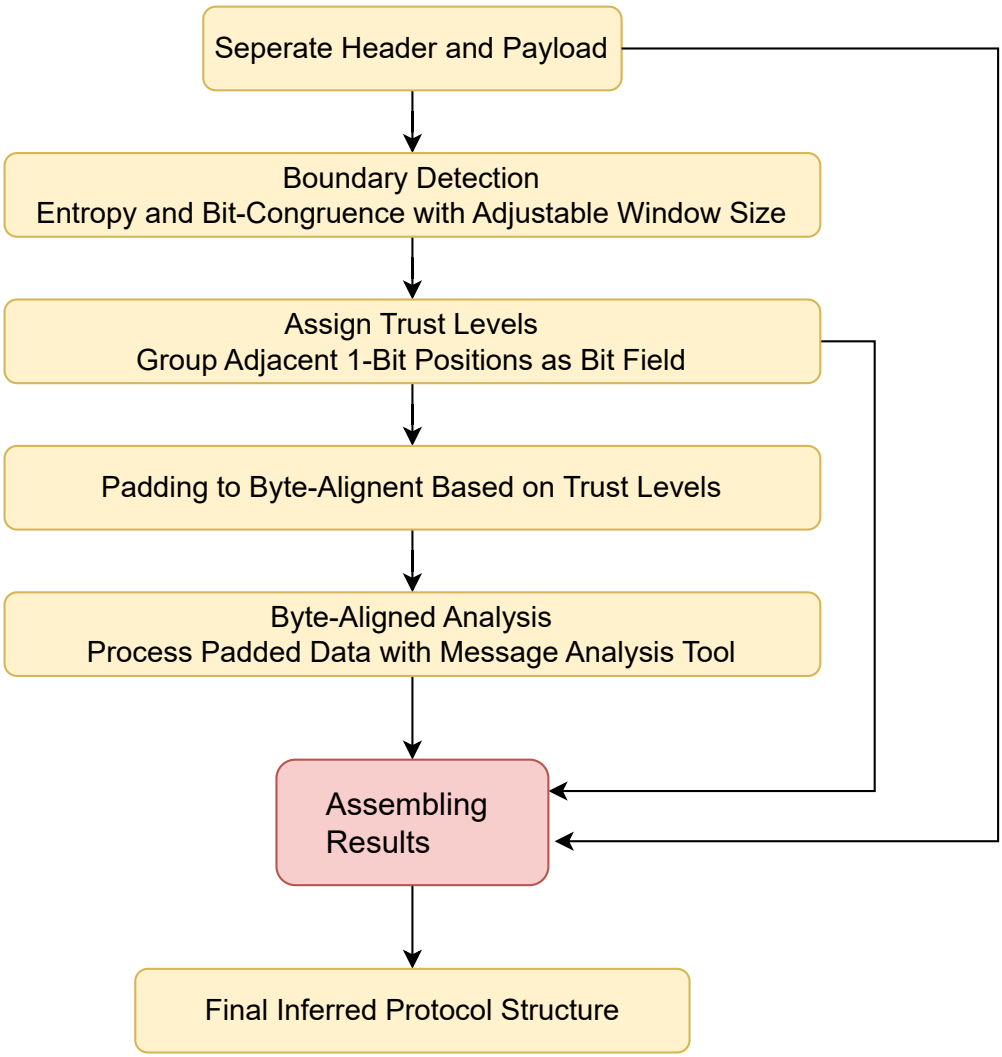


Figure 2.1: The flowchart of BBAE.

the message into progressively smaller units, starting with coarse-grained byte-level segments and narrowing down to half-byte, multi-bit, and ultimately single-bit segments. This hierarchical segmentation ensures that every possible bit position is treated as a potential boundary, allowing the method to capture both large-scale structural divisions and fine-grained sub-byte variations.

To be more specific, we first need to separate the protocol header from the payload, since the structural information of the protocol is typically encoded in the header rather than in the payload. The header generally contains fields that specify control information, message length, and field organization, all of which are essential for accurately inferring the underlying format. In contrast, the payload usually carries user or application data, which does not provide direct insight into protocol structure. Therefore, isolating the header at the beginning of the analysis is a necessary prerequisite for meaningful boundary detection. Since the protocol header normally includes the header length field, we apply the Auto-ETLV approach [8] to identify the header of the unknown protocol. This method requires the input protocol to follow the ETLV format [8]. It identifies the length field, which allows us to infer the type and value fields. Consequently, we can obtain the header from the type field (maybe implicit), the length field, and the offset. Nevertheless, since not every message contains a payload, we define the header as either the minimum length of the shortest message or the combination of the type, length, and offset fields.

After we separate the header and the payload, we then apply an adjustable window and entropy analysis on the header to identify potential bit field boundaries in the protocol header.

2.2.1 Adjustable Window

The key insight behind adopting an adjustable window is to treat every bit of the protocol header as a potential boundary candidate. Instead of relying on a fixed unit size such as a byte, the adjustable window progressively refines the granularity of analysis so that boundaries at both coarse and fine levels can be systematically considered. Specifically, we begin by dividing the message into one-byte segments, then further partition it into half-byte segments, followed by 2-bit segments, and finally down to individual 1-bit segments, as illustrated in Fig. 2.2. This hierarchical segmentation strategy ensures that no possible boundary is overlooked, regardless of whether it occurs at a byte-aligned or sub-byte position.

Once the message has been segmented, we treat the boundary of each segment as a

potential field boundary. This procedure generates a comprehensive set of candidate positions covering the entire header. Importantly, the structure shown in Fig. 2.2 highlights that boundaries detected at coarser levels (e.g., one-byte or half-byte divisions) can be verified and refined by boundaries obtained from finer divisions (e.g., 2-bit or 1-bit segments). In this way, the framework introduces an internal consistency check: longer segments provide initial candidate boundaries, while shorter segments either confirm or invalidate them.

After segmentation, we calculate the randomness and similarity of these segments using entropy and bit-congruence, as detailed in Section 2.2.2. These measures are then combined through a cross-verification process (Section 2.2.3), where results from different window sizes reinforce one another. This cross-layer validation reduces spurious detections caused by noise and enhances the reliability of the inferred boundaries, ultimately enabling a more robust and fine-grained mapping of the protocol header.

2.2.2 Entropy and Similarity Analysis

Generally, a packet header consists of control fields, such as flags, and variable fields, such as length fields and value fields. Some control fields, especially in binary protocols, have limited variation and are often only a few bits long. In contrast, variable fields are typically longer and exhibit higher randomness in length. For example, most messages do not contain the same header content multiple times, except in cases of retransmission. At the boundary of different types of fields, such as the boundary between a control field and a variable field, we may observe a significant change in randomness and similarity due to the different properties of different fields. Therefore, we leverage Shannon's entropy [15] to calculate the randomness of the segments in the same position across all the messages using the following formula:

$$H(X) = -\sum_{i} P(S_i) \log_2 P(S_i), \tag{2.1}$$

where X denotes the segments at the same position across all messages in our dataset, S_i denotes the i-th distinct segment value in X, and $P(S_i)$ is the empirical probability of S_i, calculated as the number of occurrences of S_i in X divided by the size of X. A higher entropy $H(X)$ value indicates greater randomness at the position under consideration.

Entropy alone may not be sufficient to reliably identify field boundary positions, because it serves only as a probabilistic measure of randomness. While entropy can

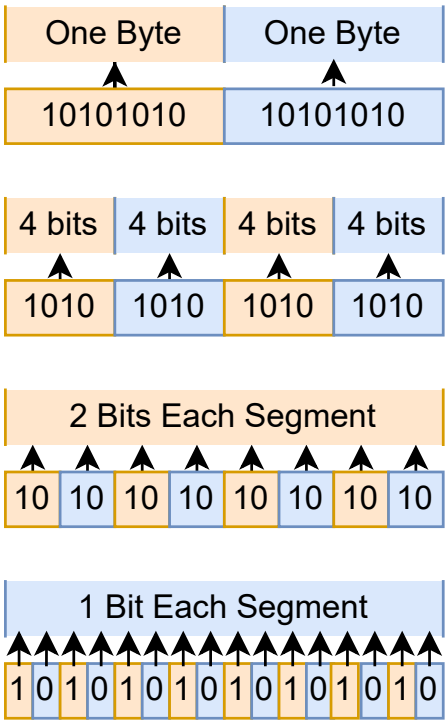


Figure 2.2: Adjustable window size: Four different window sizes are used to analyze the information within a segment to infer potential field boundaries.

highlight points where the statistical distribution of bits changes significantly, such changes do not always correspond to true field boundaries and may instead reflect noise or incidental variation in the data. To address this limitation and strengthen the robustness of boundary inference, we introduce an additional metric, similarity, to work in conjunction with entropy.

In particular, we adapt and extend the similarity calculation proposed in [10]. The original method computes bit-congruence between two adjacent bytes within a message, capturing the degree of alignment or stability at their boundaries. In our framework, this idea is generalized by measuring the average bit-congruence of segments at the same

position across all messages in the dataset. This modification enables us to exploit cross-message consistency rather than relying solely on within-message comparisons, thereby producing a more comprehensive and stable measure of similarity.

The key advantage of this approach is that it allows us to cross-verify the results of entropy analysis. Entropy identifies potential transition points, while similarity evaluates whether these points are consistently supported across multiple instances of the protocol trace. Boundaries that are confirmed by both metrics are more likely to be true positives, whereas boundaries identified by entropy alone but unsupported by similarity are treated with greater caution. In this way, the integration of similarity with entropy reduces uncertainty, improves reliability, and ultimately enhances the overall accuracy of field boundary detection.

To be more specific, when we consider a segment S of length L (i.e., the bit sequence of the same position in the header), we want to check the similarity of this segment across all the messages. For this, we first compare the same segment positions between every pair of messages, as shown in Equation (2.2):

$$BC(i,j) = \frac{cong(S_i, S_j)}{L}, \quad \text{for } 1 \leq i < j \leq N, \tag{2.2}$$

where $cong(S_i, S_j) = \left|\{0 \leq k < L : S_i^k = S_j^k\}\right|$, and S_i^k, S_j^k denote the k-th bit of the segment in message i and message j, respectively.

Then, we average the similarities of all pairs across the whole dataset using the following equation:

$$\overline{BC} = \frac{2}{N(N-1)} \sum_{1 \leq i < j \leq N} BC(i,j) \tag{2.3}$$

where N denotes the total number of messages in the dataset. Note that the calculation of $\overline{BC}$ is for the current segment S. By moving the window to a different segment and calculating its $\overline{BC}$, we can observe the similarity changes at different segment positions.

2.2.3 Cross Verification

Considering that every position in the header may serve as a potential boundary, we systematically apply our analysis across the entire message space without excluding any bit a priori. To achieve this, we employ four different window sizes, ranging from coarse byte-level partitions down to single-bit divisions, thereby ensuring that the coverage extends to every individual bit of the messages. For each segment generated under these

window sizes, we evaluate its statistical and structural properties using two complementary metrics: 1) Shannon entropy for measuring the randomness and 2) bit-congruence for assessing the similarity. Together, these two metrics form a dual perspective: entropy emphasizes variability, while similarity emphasizes uniformity, and their integration offers a balanced basis for robust inference. To clarify how these two measures interact in practice, we next present a motivating example that demonstrates their combined application. This example illustrates how entropy and similarity, when applied in tandem, can effectively highlight highly likely field boundary positions, thereby providing an intuitive understanding of the rationale behind our boundary inference process.

Example 1. *Fig. 2.3 and Fig. 2.4 show the entropy and similarity changes in the TCP [6] header, which serve as motivating evidence for our boundary detection strategy. To make the differences more interpretable, we apply a linear normalization to entropy values so that they fall within the range* $(0, 1)$*. This normalization highlights relative variations and provides a clearer view of where changes occur along the header.*

From the figures, we can observe that significant changes in entropy often arise when we consider adjacent segments in the TCP header. These abrupt changes suggest potential boundaries, since they reflect shifts in the randomness of the bit distribution. However, if we were to rely solely on entropy and treat these change points directly as field boundary positions, some true boundaries would inevitably be missed, given that entropy alone cannot fully capture every structural transition.

A similar observation holds for bit-congruence. As shown in the figures, congruence values also change across adjacent segments, indicating that these positions are likely to correspond to field boundaries as well. By incorporating bit-congruence alongside entropy, we increase the likelihood of detecting boundaries that are consistently supported by multiple statistical perspectives.

To ensure comprehensive coverage, we collect all candidate boundaries detected through both entropy analysis and bit-congruence. While this approach successfully captures a wide range of potential boundaries, it also introduces the risk of a high false positive rate, since not every detected change point corresponds to a true boundary. To mitigate this problem, we apply a cross-verification process across the results from different window sizes, enabling longer segments to validate shorter ones and vice versa. This layered checking mechanism filters out spurious detections while retaining true positives.

Finally, we implement this cross-verification using a credit-based system

(Algorithm 1). Under this scheme, boundaries accumulate "credits" when they are detected consistently across multiple windows or metrics, while isolated detections with little support are down-weighted. In this way, the credit-based process systematically reduces false positives and yields a more reliable set of final candidate boundaries.

Note that we need a criterion to check the "significant" cha nges in entropy and similarity. For this, we use percentage change to measure the significance. The percentage change is calculated by

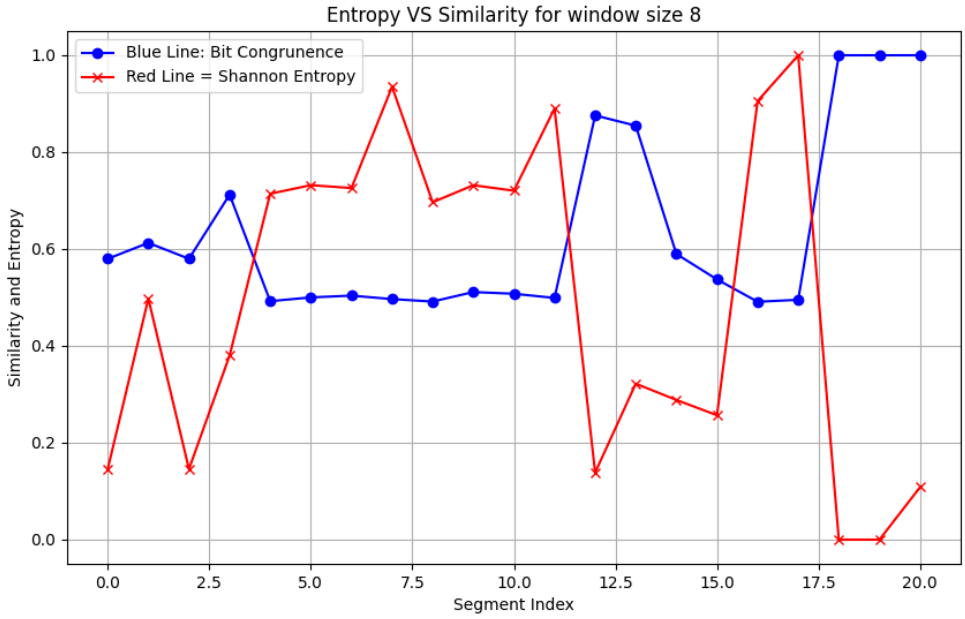
$$PC(i) = \frac{|K_i - K_{i-1}|}{K_{i-1}}, \tag{2.4}$$

where K_i and K_{i-1} denote the entropy (or similarity) of segment position i and the entropy of segment position $i-1$ in the header, respectively. If the percentage change is above a certain threshold α, we consider this change as significant. We defer the decision of the suitable α value to Section 3.1.

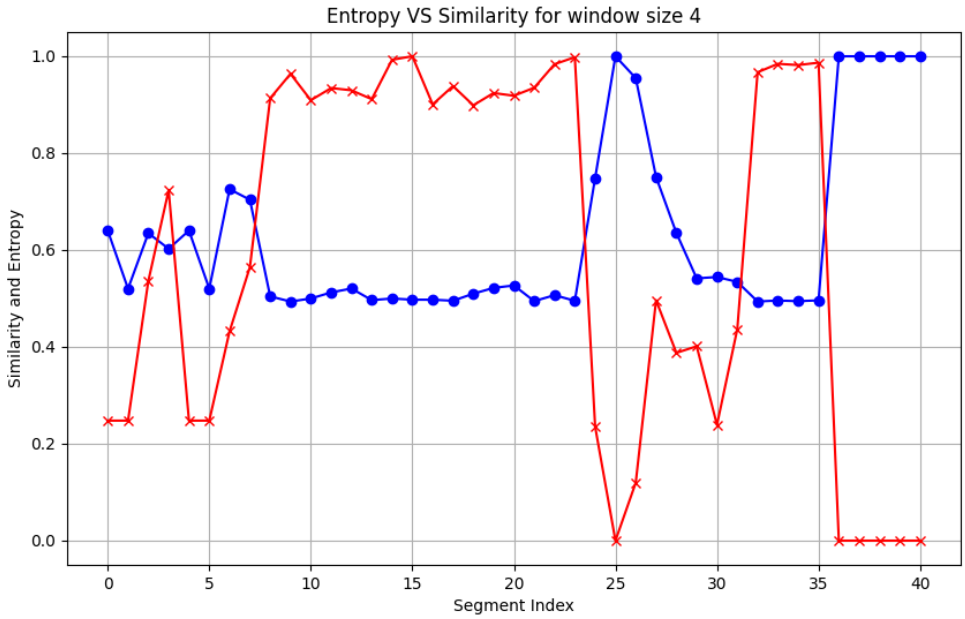
For each window size, we have so far obtained a set of potential boundary points. Next, we use a credit-based system to reduce the uncertainty. The main idea is to count the number of appearances of a specific point in different sets. For instance, the position 16 appears in the set obtained with window size 8 and the set with window size 4, so the credit of this position is 2. The position 32 appears in all the sets, so the credit of this position is 4. We use a global dictionary to assign credits for each boundary candidate. The pseudocode is shown in Algorthm 1.

A key limitation of the adjustable window mechanism arises when using the 1-bit window size. At this finest level of granularity, the results cannot be cross-verified against smaller segments, as no finer segmentation exists beyond individual bits. This lack of cross-validation introduces the risk of false positives, since random variations or noise may cause individual bits to be mistakenly identified as field boundaries.

To address this issue, we employ a heuristic method designed specifically to mitigate spurious detections at the bit level. The rule is straightforward: if an isolated bit is detected as a potential field boundary without any supporting evidence from adjacent positions, we do not treat it as a valid field. Such isolated detections are more likely to represent noise than meaningful structural information. In contrast, if three or more consecutive bit-level positions are detected as potential boundaries, we regard this pattern as a strong indicator of a bit-length control field. This interpretation is consistent with the design of many real-world protocols, where control fields, such as flag fields in TCP and MQTT [1]. They are often implemented as clusters of adjacent single-bit flags.

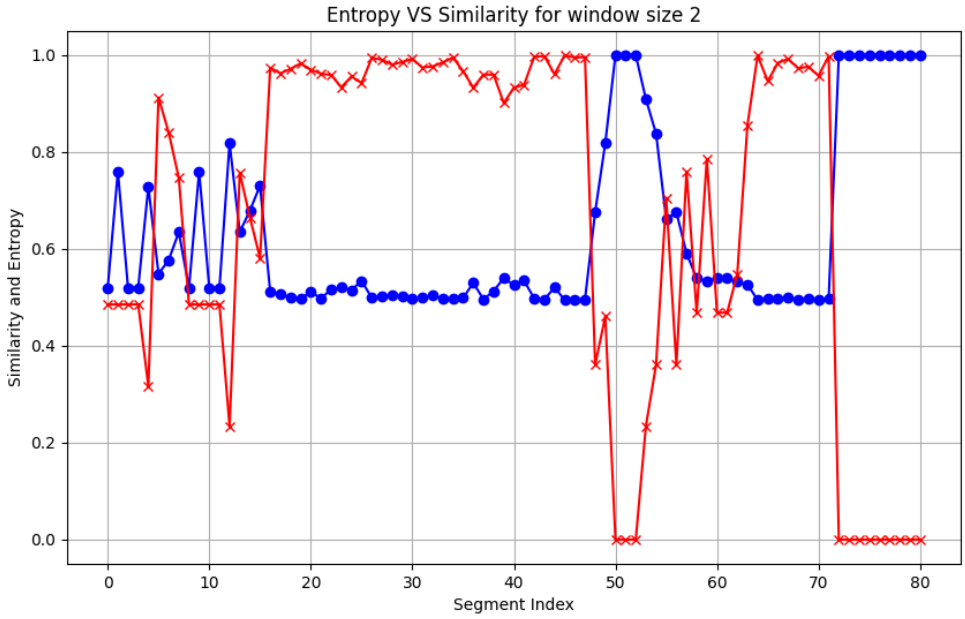


(a) Window size = 8

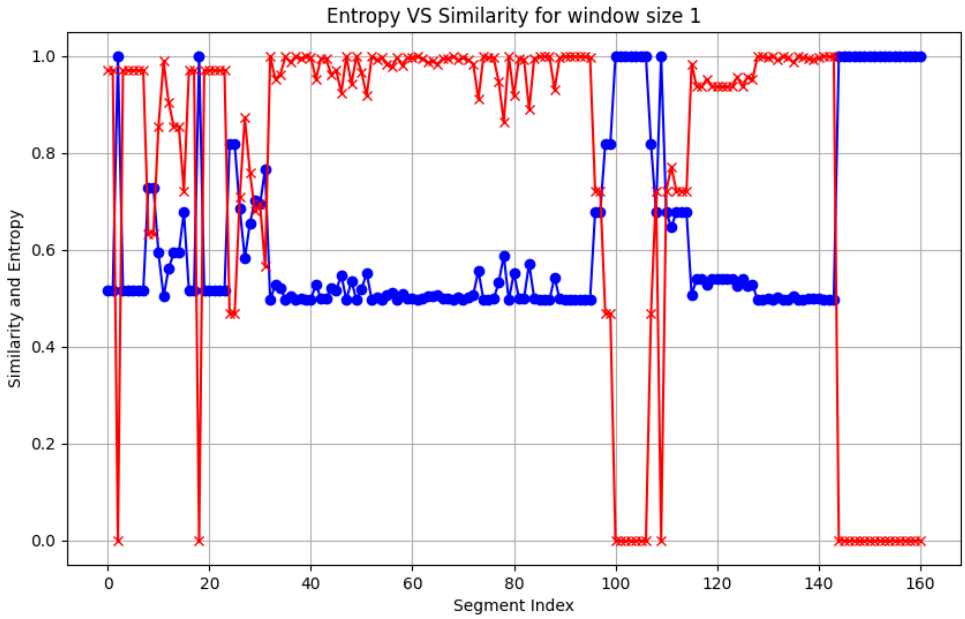


(b) Window size = 4

Figure 2.3: Entropy and similarity for TCP headers (larger windows). The blue line denotes similarity, and the red line denotes entropy.



(a) Window size = 2



(b) Window size = 1

Figure 2.4: Entropy and similarity for TCP headers (smaller windows). The blue line denotes similarity, and the red line denotes entropy.

Algorithm 1 Credit Assignments with Multiple Window Sizes

INPUT: Four lists store the relative changes produced by entropy and similarity calculations, each corresponding to one of the four different window sizes: *eight, four, two, one*.
OUTPUT: A dictionary that uses the positions where entropy and bit-congruence exhibit significant changes as keys, and the credit value of each position as the corresponding value.
Initialize: credit_dict $\leftarrow \{\}$ {Dictionary to store positions and credits}
Set: Threshold $= \alpha$
entire_lists = []
Append the list *eight, four, two, one* to entire_lists
for list in entire_list **do**
  for relative_change in list **do**
    if relative_change $\geq$ Threshold **then**
      if the relative position is not in credit_dict **then**
        Add the relative bit position into credit_dict
      else
        Add credit to the existing position in credit_dict
      end if
    end if
  end for
end for
Return: credit_dict

By applying this heuristic, we balance the need for sensitivity in detecting fine-grained fields with the requirement for robustness against false positives. The approach ensures that meaningful bit-level fields are preserved while random, isolated detections are systematically discarded. This refinement strengthens the reliability of BBAE's boundary inference, particularly in scenarios where sub-byte fields play a critical role in protocol semantics.

2.2.4 Step 2: Padding

At the end of Step 1, the boundary inference process produces a set of candidate positions by applying the credit-based verification and heuristic rules. Specifically, all positions that accumulate two or more credits through cross-verification, together with all instances of three or more consecutive 1-bit positions, are retained as candidate field boundaries. These positions represent the most promising structural divisions detected at the bit level. Importantly, such potential boundaries allow us to identify new fields that cannot be captured by state-of-the-art byte-aligned analysis tools, since these tools inherently overlook sub-byte structures.

To make these candidate boundaries compatible with existing PRE tools, we introduce a reversible padding algorithm that extends sub-byte fields identified in Step 1 to full-byte length. The padding process is simple yet effective: for each bit-length field, we insert zeros at the beginning of the field, thereby shifting it to the nearest byte boundary. This transformation enables subsequent byte-aligned analysis while leaving the semantic value of the original field unchanged. To guarantee reversibility, we record the positions of all padded bits so that they can later be removed, restoring the true bit-level representation once the analysis is complete.

The design of the padding algorithm follows two key principles:

- Preservation of original structure: The protocol's inherent semantics remain unchanged.
- Reversibility: The padded structure can be reverted to its original form after analysis.

2.2.5 Step 3: Byte-Aligned Analysis

Once the protocol fields have been padded to achieve byte alignment, we are able to leverage existing byte-oriented PRE tools to conduct semantic analysis and to verify the

candidate boundaries detected in earlier steps. This stage capitalizes on the strengths of established tools that have been optimized for byte-level analysis, thereby enhancing both the reliability and interpretability of the inferred protocol structure. In this way, BBAE transforms otherwise inaccessible sub-byte boundaries into analyzable byte-aligned segments without compromising their underlying semantics.

A significant strength of BBAE lies in its modular design, which allows seamless integration with a wide variety of byte-oriented PRE tools. Rather than being tied to a single implementation, BBAE functions as a flexible framework where different tools can be interchanged depending on the analysis context. This flexibility is particularly important in practice, as no single PRE tool is universally optimal for all protocols or datasets. By design, BBAE accommodates multiple options, allowing users to take advantage of state-of-the-art (SOTA) methods according to their own requirements, whether prioritizing speed, accuracy, or interpretability.

Through this integration, BBAE not only extends the applicability of byte-oriented tools to protocols with bit-level boundaries but also demonstrates its value as a general purpose framework. By bridging bit-level detection with byte-level validation, BBAE ensures that its results remain robust while remaining compatible with the best available analysis techniques in the PRE domain.

For this study, we adopt BinaryInferno [3] as a representative example because of its demonstrated state-of-the-art (SOTA) performance in byte-aligned protocol reverse engineering. BinaryInferno has been widely recognized for its ability to handle a broad range of protocols efficiently and accurately, making it a strong baseline for evaluating the effectiveness of BBAE's integration with byte-oriented tools.

BinaryInferno excels in detecting a variety of structural elements commonly found in protocol headers, such as control fields, timestamps, and length fields. It achieves this by leveraging a combination of atomic detectors that identify low-level statistical regularities and pattern-based analysis that captures higher-level structural cues. Together, these techniques allow BinaryInferno to provide robust detection results when the protocol fields conform to byte-aligned boundaries, offering high accuracy in standard PRE scenarios.

However, BinaryInferno is inherently limited by its granularity of analysis: the byte is treated as the smallest indivisible unit. As a result, any field that occupies fewer than eight bits, such as flag bits or other compact control indicators, cannot be inferred using BinaryInferno alone. These sub-byte fields are simply invisible to its detectors, creating blind spots in the reconstructed protocol structure.

By introducing the reversible padding mechanism in BBAE, we are able to bridge this gap. Sub-byte fields detected in the earlier bit-level analysis are padded to form full-byte segments, which are then fed into BinaryInferno. This transformation allows BinaryInferno to analyze fields that would otherwise remain undetected, effectively extending its applicability beyond byte-aligned boundaries. Once BinaryInferno completes its analysis, the padding is removed to restore the original structure, ensuring that the true bit-level semantics are preserved. Through this integration, BBAE not only enhances the capability of BinaryInferno but also demonstrates its role as a flexible framework for enabling byte-oriented tools to operate in contexts traditionally restricted to bit-oriented analysis.

Through this seamless integration, BBAE demonstrates its ability to bridge bit-oriented and byte-oriented analysis techniques in a unified framework. The strength of this integration lies in the fact that the semantic interpretations produced by the byte-aligned analysis are not lost during the reverse transformation. Once the padding removal process is applied, the protocol representation is restored to its original bit-level format, but it retains all the structural insights obtained from the intermediate byte-level analysis. In this way, BBAE guarantees that the reconstructed bit-level view of the protocol incorporates both the precision of bit-level detection and the semantic validation of byte-level inference. This design not only preserves interpretability but also reinforces the reliability of the final output. The resulting boundary map is both accurate and semantically consistent, reflecting the strengths of two complementary paradigms. Hence, BBAE exemplifies a practical and effective pathway for extending the reach of existing byte-oriented PRE tools into domains that demand bit-level analysis, without sacrificing either accuracy or meaning.

2.2.6 Step 4: Ensembling

To enhance both accuracy and robustness, BBAE performs an ensembling step in which the boundary detections obtained from Step 1 (bit-level entropy and similarity analysis) and Step 3 (byte-aligned validation after padding) are systematically merged. This procedure consolidates evidence from multiple perspectives, ensuring that boundaries supported by different analyses are more likely to be retained as true positives.

The purpose of this ensembling process is to guarantee that the strengths of different analyses are leveraged effectively. Step 1 provides comprehensive coverage and high sensitivity to potential field transitions, particularly for sub-byte boundaries that cannot be

detected otherwise. Step 3, in contrast, introduces the reliability of byte-oriented validation, reducing false positives and confirming structural elements that align with conventional PRE tools. By integrating these complementary outputs, BBAE produces a final boundary map that is both fine-grained and semantically reliable.

2.2.7 BBAE in Practice: An Illustrating Example

To better appreciate the power of BBAE, we provide a step-by-step example that illustrates the entire workflow of the proposed method. This example focuses on the analysis of the TCP header, a representative binary protocol that contains both byte-aligned fields and sub-byte fields such as control flags. By walking through the process in detail, we demonstrate how BBAE systematically applies entropy analysis and bit-congruence with adjustable window sizes, generates candidate boundary positions, and then employs the reversible padding mechanism to convert sub-byte boundaries into byte-aligned segments. The padded data is subsequently reanalyzed using a state-of-the-art byte-oriented PRE tool, which validates the initial detections and reveals additional boundaries that were previously inaccessible.

First, we attempt to analyze the TCP header using BinaryInferno, a state-of-the-art byte-oriented protocol reverse engineering tool. The result of this analysis is presented in Fig. 2.5, which clearly illustrates the limitations of relying solely on byte-aligned assumptions. Specifically, the 4-bit length field, which occupies only half of a byte, and the 1-bit control flags (e.g., SYN, ACK, PSH), which are scattered across individual bits, cannot be detected by BinaryInferno. This failure occurs because the tool processes messages at the granularity of full-byte segments, meaning that any field smaller than a byte is effectively invisible to its detectors.

Next, we show how BBAE finds out the TCP field boundaries. BBAE's adjustable window mechanism is applied to the TCP header. The entropy and bit-congruence calculations reveal boundary candidates across multiple segment sizes. Specifically:

- Sub-byte analysis detects precise boundaries for control flags and short fields.
- Using BBAE's credit assignment mechanism, boundaries that receive one or more credits are designated as boundary candidates.
- When three or more adjacent bit-level positions are identified as boundaries, they are assumed to form bit-length fields, as a one-bit window size cannot be verified by other window sizes.

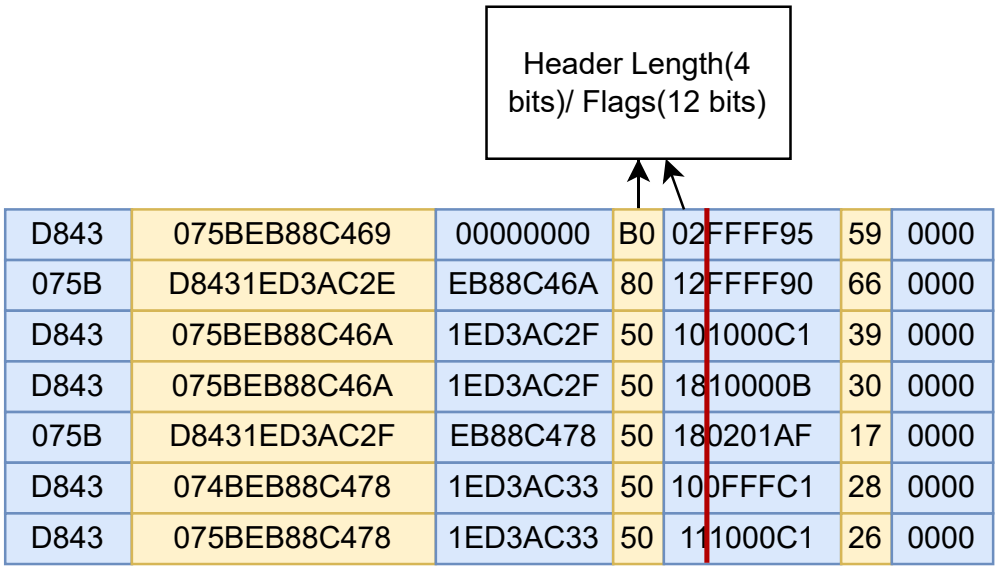


Figure 2.5: Analysis of TCP header with BinaryInferno: The messages are in hexadecimal format, where every two characters constitute a byte. The boundaries returned from BinaryInferno are incorrect. For instance, the TCP header length field starts at the 96th bit and ends at the 100th bit, and BinaryInferno cannot detect this field because it assumes each segment is at least one byte long.

In the TCP example, the adjustable window mechanism identifies a set of boundary candidates at the positions [0, 16, 32, 96, 100, 108, 110, 128, 144], which effectively covers both byte-level and sub-byte divisions within the header. Notably, the positions [108, 109, 110] correspond to three consecutive bit-level boundaries. According to our heuristic, such consecutive detections indicate the presence of a compact control field, such as the flag bits in the TCP header, which are known to occupy single-bit positions. This observation highlights the ability of BBAE to capture fine-grained field structures that conventional byte-oriented approaches inherently overlook.

After obtaining these candidate boundaries, we apply the padding mechanism to extend the sub-byte fields into full byte-aligned segments. This transformation preserves the original semantics while enabling compatibility with existing byte-oriented tools. We then reprocess the padded data using BinaryInferno, which allows us to validate the detected boundaries and, in particular, to successfully identify the 4-bit length field and extract its semantic description. Once the analysis is complete, the padded bits are removed to restore the true bit-level representation, and the verified boundaries are incorporated into the final candidate list. This step-by-step process ensures both accuracy and reversibility.

The final set of field boundaries inferred by BBAE for the TCP header is presented at the bottom of Fig. 3.4. These results demonstrate how BBAE refines raw candidate positions through entropy and similarity analysis, leverages padding to enable byte-aligned validation, and ultimately produces a boundary map that closely matches the ground truth. This example underscores the effectiveness of the framework in overcoming the limitations of existing byte-oriented tools and illustrates its capability to reveal previously undetectable bit-level fields in binary protocols

Chapter 3

Evaluation Results

We demonstrate the effectiveness of BBAE by applying it to infer the field boundaries of two representative binary protocols: TCP and MQTT. We capture TCP and MQTT traces at the client side using Wireshark at random times. The TCP trace includes 183 packets, and the MQTT trace includes 801 packets. In addition, the dataset used in our evaluation is intentionally moderate in size. BBAE does not rely on large volumes of traffic but instead requires sufficient diversity across messages to expose variations in entropy and similarity. Even with a relatively small set of traces, as long as the captured packets exhibit diverse field values, BBAE is able to reliably infer field boundaries. This ability to operate effectively on small yet diverse datasets highlights a key advantage of our approach, as it reduces the dependence on extensive traffic collection while remaining practical for real-world analysis scenarios.

TCP is particularly suitable for evaluation because its header contains a mixture of byte-aligned fields (e.g., source and destination ports and sequence number) and sub-byte fields (e.g., header length and control flags). This hybrid field layout poses a well-known challenge for most existing PRE tools, which typically assume byte-aligned boundaries and therefore struggle to accurately infer compact bit-level fields. Consequently, TCP serves as a representative example of a protocol with mixed field structures.

In contrast, MQTT extensively employs sub-byte fields in its fixed header, such as control flags and quality-of-service indicators, making it a typical example of a predominantly bit-oriented protocol. These compact field designs further expose the limitations of byte-oriented PRE approaches and provide a clear test case for evaluating BBAE's ability to detect and analyze sub-byte structures.

Together, TCP and MQTT span both hybrid and highly compact protocol designs, allowing us to comprehensively demonstrate the generality and robustness of BBAE.

Moreover, both protocols are widely deployed and well understood, making them suitable benchmarks for evaluating protocol reverse engineering techniques and for clearly illustrating the practical impact of our approach. [1].

3.1 How to Select a Suitable Threshold α?

Setting a suitable α value is generally determined on the basis of empirical observation and analysis. As illustrated in Figs. 2.3, 2.4, 3.5, and 3.6, the entropy and similarity curves for the TCP and MQTT headers exhibit clear fluctuations across segment boundaries. These figures provide direct evidence of how entropy and similarity behave under different window sizes, and they serve as the foundation for selecting a reasonable threshold.

From these observations, we find that an appropriate choice for the threshold α in Algorithm 1 is 50%. This value is supported by the fact that the majority of percentage changes, as calculated using Eqn. (2.4), exceed this threshold in both the TCP and MQTT cases. By adopting $\alpha = 50\%$, we ensure that only substantial shifts in entropy or similarity are considered significant, while smaller fluctuations that may result from noise are effectively filtered out. This setting strikes a balance between sensitivity and robustness, allowing the algorithm to capture meaningful boundary transitions without introducing excessive false positives. Importantly, this threshold is not tuned to a specific protocol or dataset size, but reflects a consistent separation between meaningful boundary induced changes and minor fluctuations observed across different window sizes and protocol structures.

Accordingly, we adopt this α value in the remainder of the thesis, and all subsequent evaluations and comparisons are conducted under this configuration. This consistent choice of parameter ensures comparability across experiments and provides a stable foundation for assessing the performance of BBAE in detecting field boundaries for binary protocols.

3.2 Evaluation Results

According to the specifications of TCP, the TCP header is defined to contain a total of 15 distinct fields[2].This design results in 15 boundary positions, excluding the initial position

[1]The datasets and the source code of BBAE are available at https://github.com/HarshbrownSs/BBAE.git

[2]Each flag in the flag field is considered a separate field in the context of protocol analysis.

at 0. These boundaries reflect the division of the header into its constituent components, including control information, addressing details, and status indicators, all of which must be accurately identified in order to reconstruct the complete protocol structure.

Following the specification of MQTT[3], the header under consideration consists of 5 fields, which correspond to 5 boundary positions when position 0 is excluded. Compared to TCP, the MQTT header is more compact, yet its boundaries are equally critical for correctly interpreting message semantics. By establishing these ground-truth boundaries from the protocol specifications, we are able to provide a clear baseline against which the effectiveness of inference methods can be measured.

For evaluation, we adopt standard classification metrics to compare inferred boundaries with the specification-defined ground truth. Specifically, any inferred boundary that aligns with a true boundary position is classified as a true positive (TP). Conversely, if an inferred boundary does not match any actual specification-defined boundary, it is regarded as a false positive (FP). Finally, when a true boundary specified in the protocol is not detected by the inference method, it is counted as a false negative (FN). This framework of TP, FP, and FN provides a precise and interpretable means of quantifying detection performance, enabling consistent comparisons across different tools and protocols.

We use the widely adopted metrics of precision, recall, and F1-score [3] to evaluate the performance of field boundary detection. These metrics are commonly employed in information retrieval and classification tasks, and they provide a standardized way to assess the effectiveness of inference methods.

Specifically, precision is calculated as the ratio of true positives (TP) to the total number of inferred boundaries $(TP + FP)$. This metric quantifies the proportion of correctly identified field boundaries among all detected boundaries, thereby reflecting the accuracy of the predictions. A higher precision value indicates that the method produces fewer false positives and is more reliable in its boundary identification.

In contrast, recall focuses on the ability of a method to capture the complete set of true boundaries. It is defined as the ratio of true positives (TP) to the total number of true field boundaries $(TP + FN)$. A higher recall value signifies that fewer true boundaries are missed during the inference process.

Finally, the F1-score is defined as the harmonic mean of precision and recall [3]. This measure balances the trade-off between precision and recall, providing a single, comprehensive indicator of overall performance. By combining both aspects, the F1-score

[3]Different MQTT versions have different header structures. The MQTT version under our study is 3.1.1.

ensures that neither precision nor recall dominates the evaluation, making it a particularly useful metric for comparing different PRE tools under the same experimental conditions.

For each protocol, boundary detection is performed once on the entire dataset, producing a single set of inferred boundary positions for the protocol header. This set is then compared against the complete set of ground truth boundaries. Therefore, TP, FP, and FN are computed based on one global comparison per protocol rather than per packet. This evaluation strategy is appropriate because BBAE infers protocol structure by aggregating statistical evidence across all packets in the dataset, rather than analyzing packets independently. Because the ground truth boundaries are known and fixed for TCP and MQTT headers, repeating the same boundary comparison multiple times on identical protocol formats would not provide additional information. Instead, evaluating the inferred boundary set against the ground truth once per protocol accurately reflects the effectiveness of the method in recovering the protocol structure from network traces.

We used Auto-ETLV, BBAE, and BinaryInferno to detect the field boundaries of both TCP and MQTT, thereby enabling a direct and systematic comparison across different categories of PRE approaches. Among these methods, Auto-ETLV represents a specialized bit-oriented technique that has shown strong performance when analyzing protocols that conform to the extended type-length-value (ETLV) format [8]. In such scenarios, Auto-ETLV is particularly effective because the ETLV structure inherently encodes the location of the length field, which can then be leveraged to derive both the type field and the associated value fields.

In our evaluation, Auto-ETLV demonstrated its capability to separate the header and payload for both TCP and MQTT traces. This functionality provided us with the necessary header data, which was subsequently used as the input for both BBAE and BinaryInferno, ensuring that the comparison among the tools was conducted under consistent conditions. While Auto-ETLV proved to be accurate in identifying the length field within the header, its performance was markedly ineffective in detecting other types of fields, such as control flags or fields with sub-byte boundaries. This limitation stems from its reliance on the ETLV assumption, which does not align with the structural characteristics of many binary protocols.

The evaluation results are summarized in Table 3.1, with visual comparisons provided in Fig. 3.1 and Fig. 3.2. For MQTT, Auto-ETLV demonstrates a precision of 0.50, a recall of 0.20, and an F1-score of 0.29. These numbers indicate that while Auto-ETLV is able to correctly identify some true boundaries, it misses the majority of them, leading to very low recall. For TCP, the performance is similarly limited: the tool

identifies only 1 true positive and 1 false positive, but at the same time incurs 14 false negatives, showing its inability to capture most of the actual field boundaries.

BinaryInferno performs differently across the two protocols. For MQTT, it fails entirely, with a precision of 0, a recall of 0, and an F1-score of 0, indicating that the tool cannot detect any true boundaries when applied to this protocol. For TCP, the results are somewhat better: BinaryInferno achieves a precision of 0.67, a recall of 0.27, and an F1-score of 0.38. While the higher precision suggests that a majority of the boundaries it identifies are correct, the low recall once again highlights that many true boundaries remain undetected.

In contrast, BBAE demonstrates a clear improvement over existing tools by achieving strong results on both protocols. For TCP, BBAE obtains a precision of 1.00, a recall of 0.73, and an F1-score of 0.84. The perfect precision indicates that every detected boundary corresponds to a true boundary, with no false positives. At the same time, the recall value of 0.73 shows that BBAE successfully identifies the majority of actual field boundaries, leaving only a small fraction undetected. The combined effect of these two metrics is reflected in the high F1-score, demonstrating a strong balance between accuracy and completeness. For MQTT, BBAE achieves similarly strong performance, with a precision of 0.83, a recall of 1.00, and an F1-score of 0.91. In this case, BBAE is able to capture all true field boundaries, as shown by the perfect recall, while maintaining a high precision level. The few false positives that occur do not substantially impact the overall evaluation, as reflected in the excellent F1-score.

When compared directly against Auto-ETLV and BinaryInferno, the advantages of BBAE are evident. Its ability to achieve high precision while also maintaining strong recall demonstrates its superiority as a powerful and versatile tool for binary protocol field analysis. These results confirm that BBAE is capable of addressing the limitations of existing methods and advancing the state of the art in protocol reverse engineering.

3.3 Ablation Study of BBAE

BBAE consists of two key steps that form the foundation of its framework. Step 1 is responsible for detecting candidate field boundaries through the use of an adjustable window mechanism combined with entropy analysis and bit-congruence calculations. This step provides initial insight into potential boundary positions at both the byte and sub-byte levels, ensuring comprehensive coverage of the message structure. Step 3, in contrast, operates after the sub-byte boundaries identified in Step 1 have been processed

Table 3.1: Field Boundary Detection Results

Protocol	Method	TP	FP	FN	Precision	Recall	F1-Score
MQTT	Auto-ETLV	1	1	4	0.50	0.20	0.29
	BinaryInferno+	0	0	5	0.00	0.00	0.00
	BinaryInferno	1	0	4	1.00	0.20	0.33
	NEMESYS	1	0	4	1.00	0.20	0.33
	BBAE	**5**	**1**	**0**	**0.83**	**1.00**	**0.91**
TCP	Auto-ETLV	1	1	14	0.50	0.07	0.12
	BinaryInferno+	4	2	11	0.67	0.27	0.38
	BinaryInferno	6	3	9	0.67	0.40	0.50
	NEMESYS	6	1	9	0.86	0.40	0.55
	BBAE	**11**	**0**	**4**	**1.00**	**0.73**	**0.84**

*Precision, Recall, and F1-score are rounded to two decimal places.

through the padding mechanism, allowing the data to be aligned to byte boundaries. Once aligned, byte-oriented analysis can be applied to validate the earlier detections and refine the boundary positions, ultimately enhancing both accuracy and interpretability.

To better understand the contribution of each of these steps, we conduct an ablation study that isolates their effects and examines their impact on the overall performance of BBAE. By separately evaluating Step 1 and Step 3, we are able to assess the role of entropy-based boundary detection on the one hand, and the added value of padding-enabled byte-aligned verification on the other. This systematic comparison provides clear evidence of how the two steps complement each other, and it highlights the necessity of combining both mechanisms within a unified framework to achieve reliable and accurate boundary detection for binary protocols.

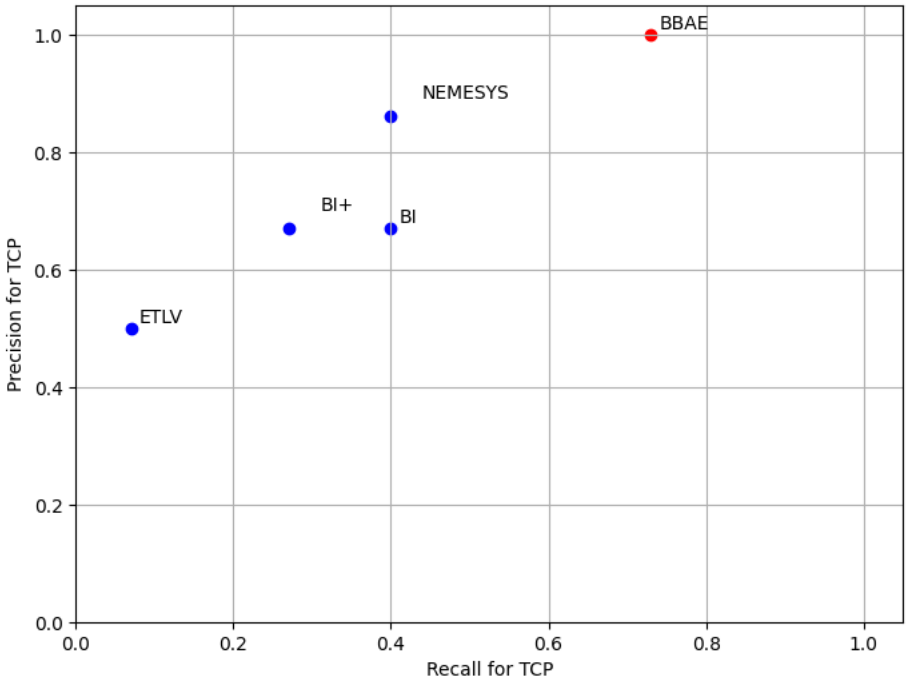
3.3.1 Contribution of Adjustable Window and Entropy Analysis

Step 1 of BBAE uses the similarity and randomness of a segment position across all the packets to infer boundaries. Figs. 2.3, 2.4, 3.5 and 3.6 show the entropy and similarity changes in TCP headers and MQTT headers, respectively. Based on these two figures, we set the threshold α in Algorithm 1 to be 50%. The red and blue arrows in Figs. 3.3 and 3.4 show the boundaries detected after this Step. The figures show that Step 1 of BBAE algorithm can correctly identify most boundaries. Nevertheless, this step still misses some boundaries in TCP, indicating the necessity of further padding-based analysis.

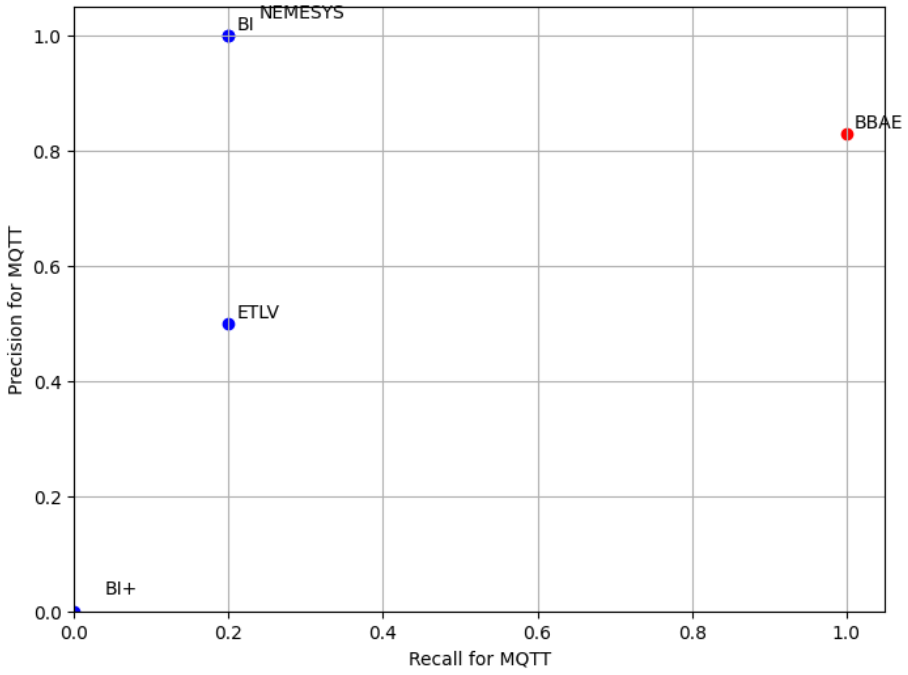
3.3.2 Contribution of Padding and Byte-Aligned Analysis

With the boundaries of the bit-length fields identified, we then apply our padding algorithm to insert additional zero bits at the beginning of each inferred field. This procedure ensures that every sub-byte segment becomes properly byte-aligned, thereby satisfying the prerequisites required by existing byte-oriented detectors. The design of this step follows two essential principles: it maintains the semantic integrity of the original protocol while guaranteeing that the inserted bits can later be reversibly removed without affecting the true field structure.

As illustrated in Fig. 2.5, fields such as the 4-bit length field and the 1-bit control flags (e.g., SYN, ACK, PSH) are inherently shorter than one byte, making them invisible to conventional byte-oriented approaches. By introducing padding, BBAE reshapes these fields into analyzable byte-aligned units, allowing tools such as BinaryInferno to operate



(a) Precision-recall for TCP.



(b) Precision-recall for MQTT.

Figure 3.1: Precision-recall for TCP and MQTT protocols.

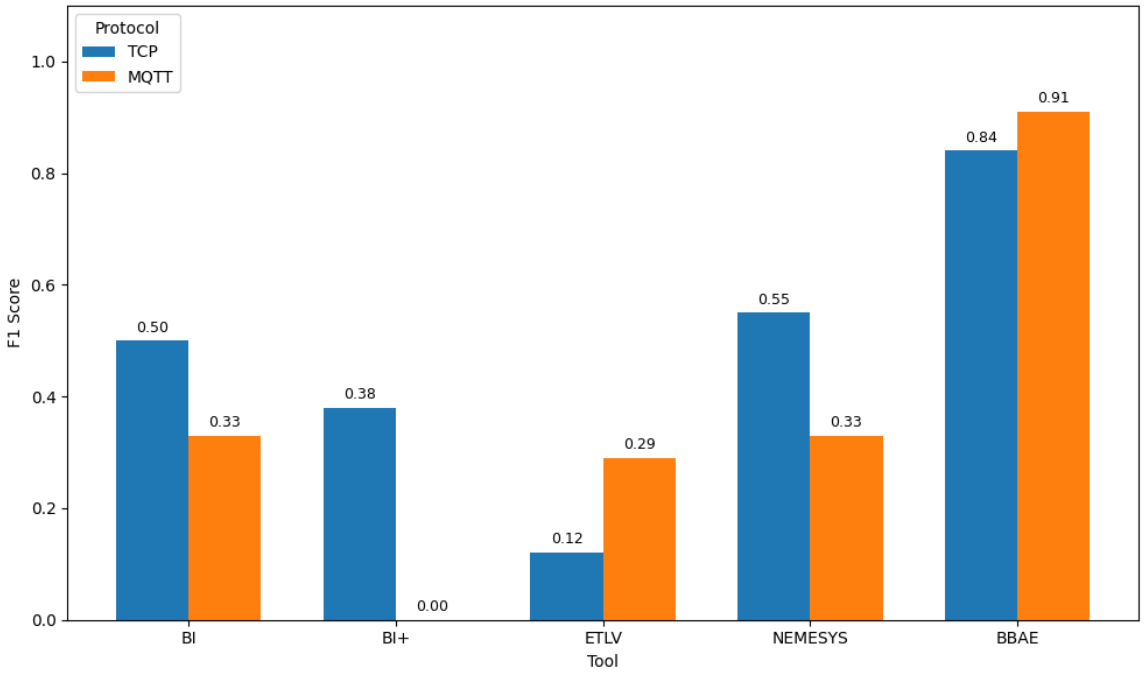


Figure 3.2: F1 score across protocols and tools.

effectively on the modified data. The green and blue arrows in Figs. 3.3 and 3.4 indicate the boundary positions detected during Step 3, after the padded data has been reprocessed.

The evaluation results confirm the importance of this step. For MQTT, Step 3 reconfirms boundaries already suggested in Step 1, such as the boundary between the message type and the DUP flag, which increases confidence in the inferred structure. For TCP, Step 3 is more impactful. It validates several boundary candidates from the initial bit-level analysis, including the boundaries between the source port and destination port, the ACK number and header length, the header length and reserved field, and the checksum and urgent pointer. In addition, Step 3 identifies a new boundary between the Sequence Number and Acknowledgement fields that was missed in Step 1.

Through this process, BBAE demonstrates that padding, when combined with byte-oriented analysis, substantially improves boundary detection in binary protocols. The ability to both confirm earlier findings and uncover additional boundaries highlights the value of Step 3 and emphasizes how the integration of padding with existing PRE tools contributes to the overall robustness of the framework

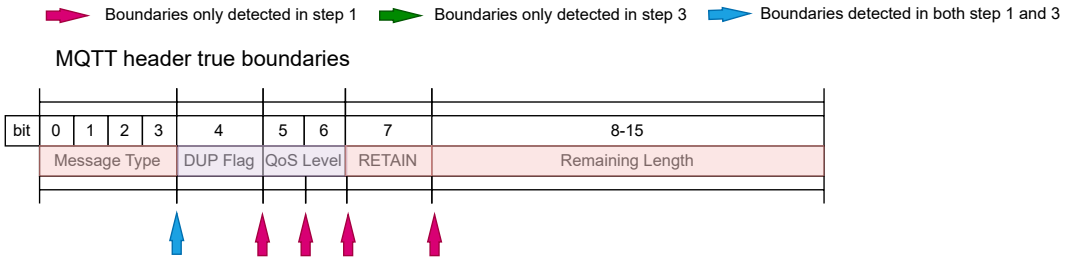


Figure 3.3: Illustration of boundary detection for MQTT at various steps of BBAE, showcasing the progressive refinement of boundaries through Step 1 and Step 3.

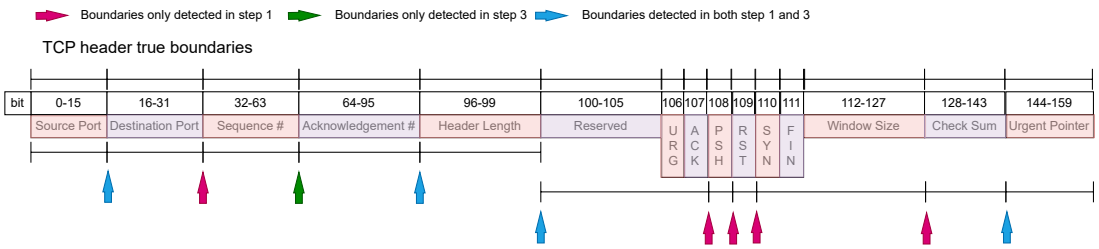
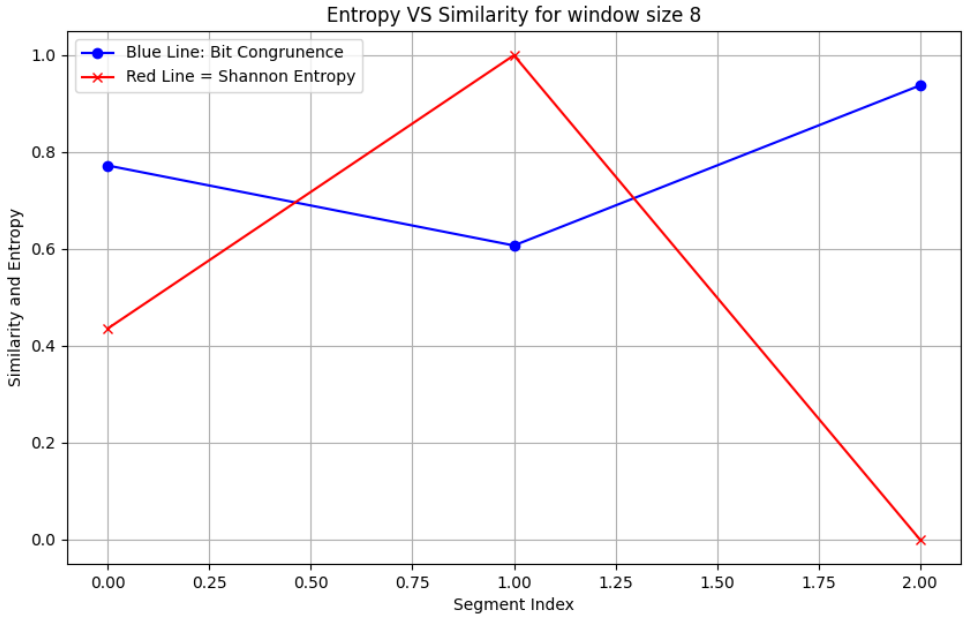
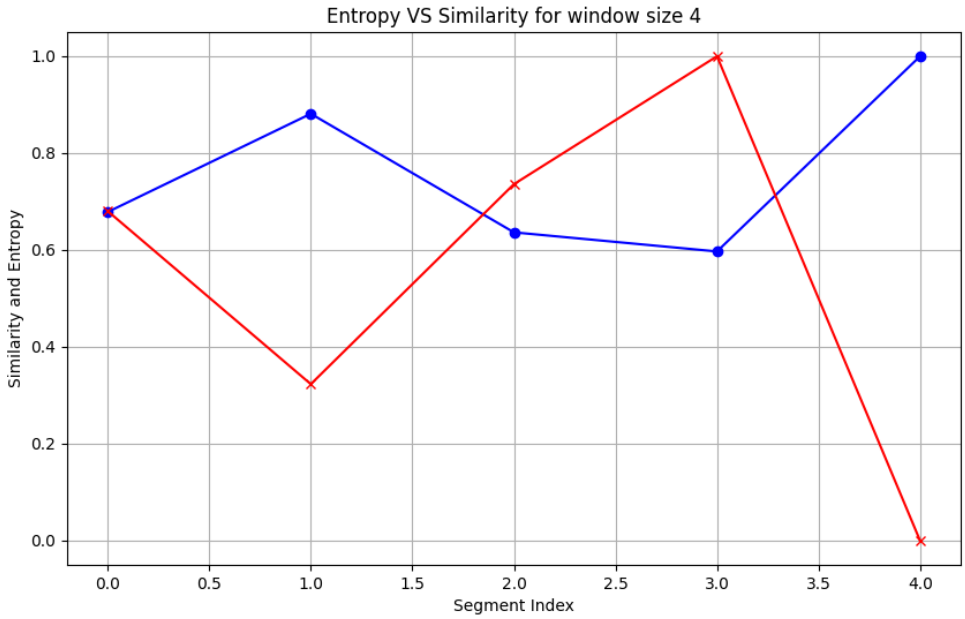


Figure 3.4: Illustration of boundary detection for TCP at various steps of BBAE, showcasing the progressive refinement of boundaries through Step 1 and Step 3.

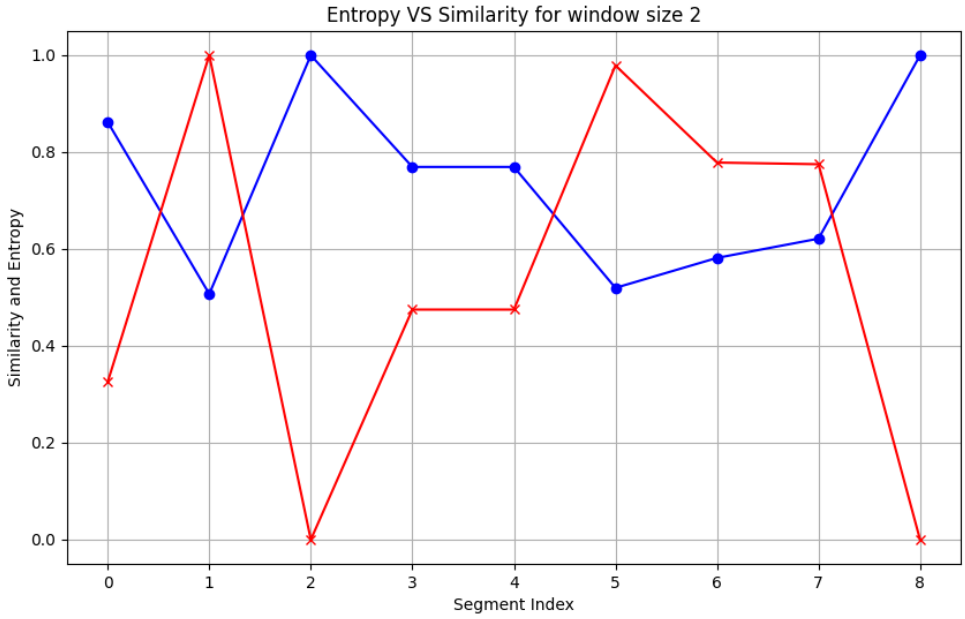


(a) The entropy and similarity of segments with a window size of 8 relative to the segment index.

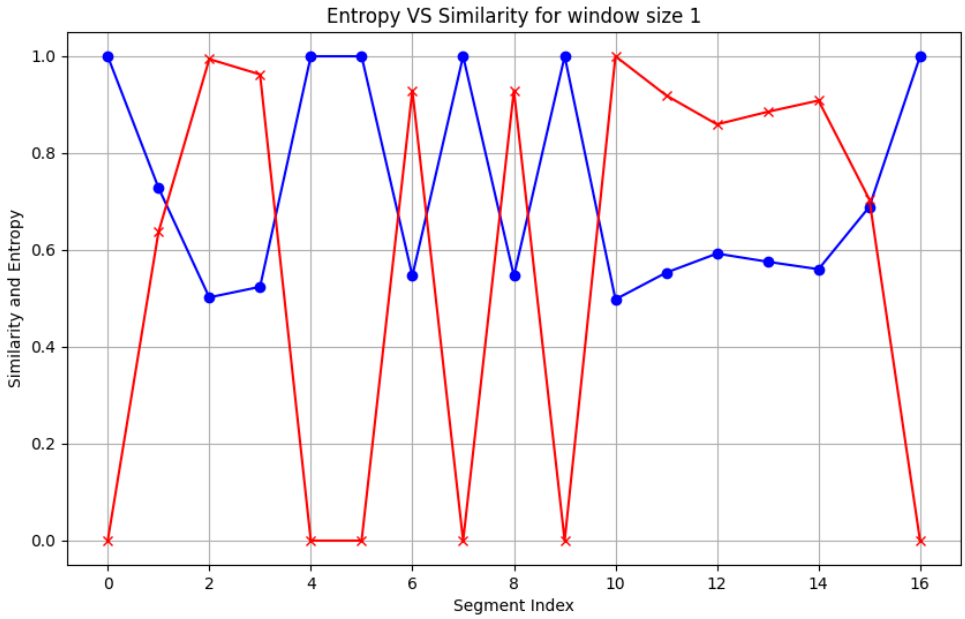


(b) The entropy and similarity of segments with a window size of 4 relative to the segment index.

Figure 3.5: Entropy and similarity for MQTT headers (larger windows: 8 and 4). Blue line denotes similarity, red line denotes entropy.



(a) The entropy and similarity of segments with a window size of 2 relative to the segment index.



(b) The entropy and similarity of segments with a window size of 1 relative to the segment index.

Figure 3.6: Entropy and similarity for MQTT headers (smaller windows: 2 and 1). Blue line denotes similarity, red line denotes entropy.

3.3.3 Further Discussion

Compared to the original version of the bit-length protocol, where BinaryInferno's detectors were unable to recognize any sub-byte fields, the application of our padding algorithm fundamentally changes the outcome. With padding in place, BinaryInferno is no longer constrained by its byte-level granularity. Instead, it is able to successfully identify fields that were previously undetectable, including those occupying only a few bits. More importantly, the tool is also capable of providing semantic descriptions for these fields once they have been aligned to byte boundaries, thereby extending its analytical power without requiring modifications to its core design.

This result highlights the transformative role of the padding mechanism within BBAE. By bridging the gap between bit-level detection and byte-level validation, the framework enables existing state-of-the-art tools to address scenarios they were not originally designed to handle. In doing so, BBAE introduces a novel and generalizable approach for handling protocols with bit-length fields, a long-standing challenge in protocol reverse engineering. The significance of this contribution can be viewed as a step from 0 to 1.

At the same time, it is important to consider whether padding may introduce negative effects. Padding can potentially affect boundary detection if it alters the statistical or semantic properties of the data used for analysis. In BBAE, however, the padding mechanism is carefully designed to minimize this risk and to constrain its impact.

First, padding is applied only after candidate boundaries have already been identified at the bit level in Step 1. As a result, the padding process does not affect the initial boundary-suggestion stage, which relies on entropy and bit-congruence computed from the original, unmodified bit sequences. Therefore, padding cannot introduce false boundaries during bit-level detection.

Second, the padding operation in BBAE is structure preserving and fully reversible. 0s are inserted solely to extend sub-byte fields to byte-aligned lengths, without modifying the original bit patterns of the fields themselves. Because both the relative ordering and the internal values of fields are preserved, the semantic meaning of each field remains unchanged. This design ensures that byte-oriented tools operate on a transformed representation that remains statistically compatible with the original protocol structure.

In our experimental evaluation, we did not observe any negative impact of padding on boundary detection accuracy.

Another practical consideration concerns encrypted network traces. BBAE, like most

trace-based PRE methods, relies on observable statistical variations in protocol fields. If the protocol payload or header is encrypted, these variations may be obscured or eliminated, which makes entropy and similarity-based boundary detection ineffective. In such cases, BBAE is not expected to directly recover meaningful field boundaries from fully encrypted traffic.

However, this limitation is not unique to BBAE and applies broadly to network trace-based PRE approaches. In practice, many protocols expose at least partially unencrypted headers for routing, control, or session management purposes, even when their payloads are encrypted. BBAE can be applied to these visible portions to infer structural boundaries. Addressing fully encrypted traffic, for example, by integrating BBAE with side-channel information, is a promising direction for future work.

Effectiveness and Reasons for False Detection: The above evaluation results demonstrate that BBAE is effective. Compared to the protocol specification of MQTT (version 3.1.1), BBAE successfully identifies all field boundaries. The only exception is a single false positive, where it splits the QoS field into two separate fields. However, strictly speaking, this may not be considered a false positive. Since MQTT packets contain different values at the QoS level, BBAE identified the high entropy changes inside this QoS field, resulting in splitting the field into two subfields. Practically, this implies BBAE can distinguish different QoS levels.

Similarly, BBAE accurately detects all field boundaries of TCP except for the Reserved and Flag fields. Upon closer examination of the packet contents, we found that this issue arises due to the facts that (1) the concatenation of (Reserved, URG, and ACK) remains nearly unchanged across all observed packets, (2) there are only several SYN and FIN packets in the dataset, which cannot cause sufficient entroy changes at the SYN and FIN flags. These limitations could be mitigated if the captured traffic were sufficiently diverse, allowing BBAE to distinguish variations within these fields. For example, forcibly corrupting the TCP server during traffic capture to introduce different URG bit values or manually altering flag field values could enable BBAE to detect these boundaries. However, we opted against such modifications since it implies that we already know the protocol structure. Instead, we only reported our evaluation results based on traffic traces captured under normal network operating conditions.

Running Time: We evaluated BBAE and other baseline methods on a laptop (Operating System: MacOS Sequoia 15.1, Memory: 8GB, CPU: Apple M2). The results demonstrate that BBAE requires approximately 4 seconds to analyze the TCP packets and around 8 seconds to process the MQTT packets. In comparison, the baseline methods

exhibit slightly faster runtimes, completing their analyses in under 2 seconds. Although this shows that BBAE introduces some additional computational overhead, the difference remains relatively small when placed in the broader context of protocol reverse engineering tasks. It is important to note that Protocol Reverse Engineering (PRE) is generally performed offline after network packet traces have been captured. As such, execution speed, while relevant, is not the primary limiting factor. Instead, accuracy and reliability of field boundary detection play a more decisive role in the utility of a method. From this perspective, the additional few seconds required by BBAE are justified by the substantial improvement in detection accuracy demonstrated in our evaluations. Moreover, a runtime measured in single-digit seconds on a standard laptop can be considered both lightweight and practical for real-world applications. This performance characteristic ensures that BBAE can be readily deployed in typical research or operational environments without the need for specialized hardware or excessive computational resources. Thus, the evaluation confirms that BBAE achieves a balanced trade-off between accuracy and efficiency, making it a feasible and effective tool for protocol analysis tasks

Chapter 4

Graphical User Interface for BBAE

To improve the usability and accessibility of BBAE, we developed a Graphical User Interface (GUI). The GUI provides an intuitive and interactive environment for conducting protocol reverse engineering tasks, thereby lowering the entry barrier for users who may not be familiar with command-line operations or programming practices. By presenting BBAE's functionality through a graphical interface, users are able to engage directly with the analysis process in a more straightforward and efficient manner.

This chapter introduces the motivation behind developing the GUI, outlines its overall architecture, and describes the key functionalities that support protocol reverse engineering with BBAE. In addition, we provide an illustrative example workflow to demonstrate how the GUI operates in practice and how it streamlines the process of analyzing binary protocols.

4.1 Motivation and Design Goals

The motivation for developing a GUI for BBAE stems from the need to improve both usability and accessibility. While the core algorithm of BBAE can be executed through command-line operations, such an interface often requires familiarity with coding environments and command-line syntax, which may not be friendly to users who lack this background. By providing a GUI, we lower the barrier and make it possible for users with diverse technical backgrounds to benefit from the analytical capabilities of BBAE.

The design goals of the GUI are therefore twofold. First, it aims to provide an intuitive and interactive environment in which users can operate BBAE without the need for command-line interaction or coding skills. Second, it seeks to preserve the full

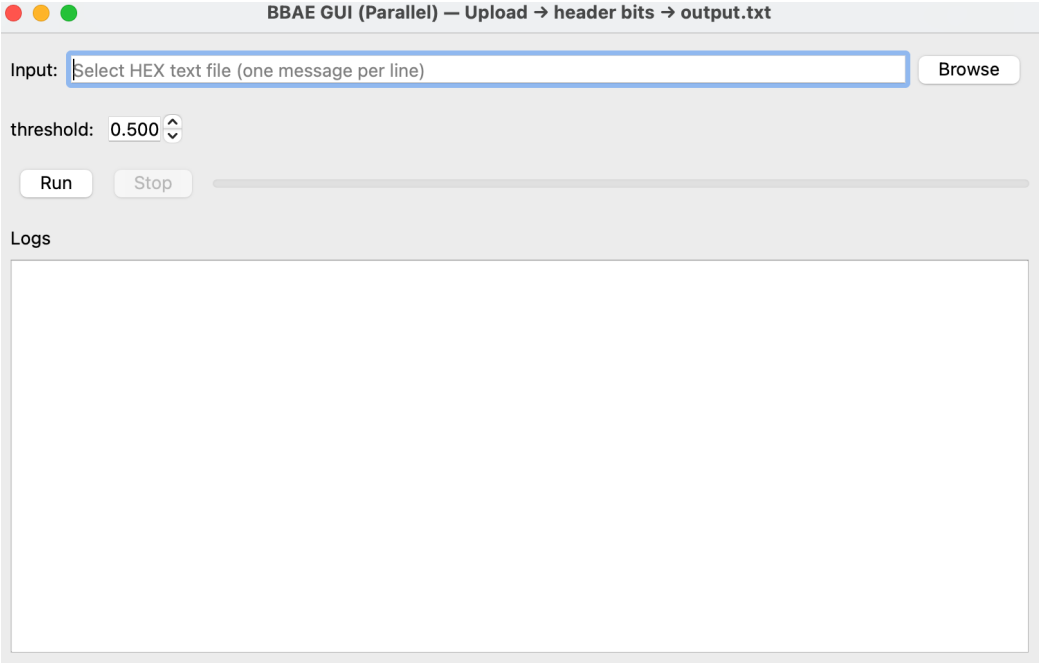


Figure 4.1: Main user interface of the BBAE GUI.

analytical power of BBAE while presenting results in a more user-friendly manner. In this way, the GUI serves as a complementary component that enhances the applicability of BBAE in both research and practical protocol analysis scenarios.

4.2 Prerequisites

Before running the GUI, the input data must be prepared by extracting the target protocol layer from the captured packets and saving it in hexadecimal format. Using `Scapy` in Python, we read the PCAP file, select the specific protocol layer for analysis, and employ Auto-ETLV to identify message headers automatically. The extracted headers are then exported in hexadecimal format, with one header per line. This preprocessing step ensures that only the relevant protocol layer is provided to BBAE for subsequent analysis.

4.3 Overview and Key Functionalities

A screenshot of the main interface is shown in Fig. 4.1. This figure illustrates the layout of the GUI, including the input panel for file selection, the parameter configuration panel, and the results area where intermediate and final outputs are displayed.

The GUI provides a set of functionalities that make protocol reverse engineering with BBAE more intuitive and user-friendly. The interface is designed to guide users through the main steps of analysis in an interactive manner. The core functionalities include:

- **File Management:** Users can open protocol trace files and manage their inputs directly from the GUI without requiring command-line interaction.
- **Parameter Configuration:** The GUI allows users to set analysis parameters, such as thresholds and header length, through simple input fields. Note that the header length is obtained based on the output of Auto-ETLV from Step 1 in the workflow shown in Fig. 2.1; a detailed explanation of this process is given in Section 2.2 (*Step 1: Boundary Suggestion*). This ensures that subsequent boundary detection is applied specifically to the header rather than the entire message.
- **Step-by-Step Output:** The GUI simultaneously outputs the results of all analysis steps in the form of lists. For each step, the detected boundary positions are displayed, allowing users to examine the intermediate outcomes in detail. In addition, the final reconstructed boundary map is presented alongside these intermediate results. By showing all steps together, the GUI makes it clear how boundaries evolve throughout the process and enables users to compare different stages of analysis in a single view.
- **Interactive Adjustment:** Since the GUI displays the outputs of all steps, users can directly compare intermediate and final results within a single view. This feature allows users to iteratively adjust analysis parameters, re-run the process, and refine results, thereby supporting trial-and-error exploration of protocol structures in a more systematic and transparent manner.

4.4 Example Workflow

This section demonstrates how to operate the BBAE GUI for analyzing a packet trace. After launching the application, we first click the *Browse* button in the input panel to select the packet file as the input, as illustrated in Fig. 4.2.

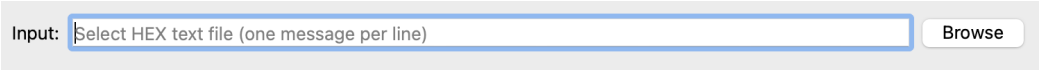


Figure 4.2: Input panel of the BBAE GUI. Users can select a packet trace file by clicking the *Browse* button before running the analysis.

Next, we configure the parameters in the parameter panel. The user needs to specify the *threshold* parameter, which may be left at its default value for an initial run or adjusted according to specific needs, as illustrated in Fig. 4.3. After clicking the *Run* button, the analysis process begins automatically.

Figure 4.3: Input panel of the BBAE GUI.

Finally, the statistical results are displayed in the log panel, as illustrated in Fig. 4.4. This panel presents the outputs of all steps together with the final reconstructed boundaries, allowing the user to verify the progression of the analysis and inspect the comprehensive results in a single view.

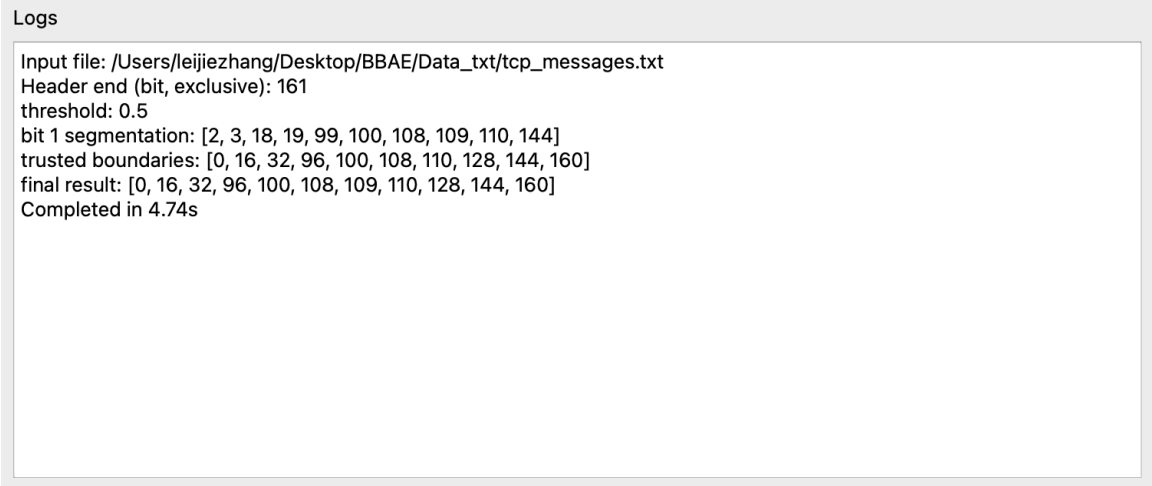


Figure 4.4: Log panel of the BBAE GUI.

By combining the 1-bit long segmentation with the trusted boundaries, as described in Section 2.2.6, we obtain the final Step 1 result, as shown in Fig. 4.4. This result corresponds to the output of Section 2.2 (*Step 1: Boundary Suggestion*). The application then generates a padded `.txt` file, which can be processed by a byte-aligned tool such as BinaryInferno. By combining the Step 1 result from BBAE with the output of the byte-aligned tool, we obtain the final result, as reported in Table 3.1.

Chapter 5

Conclusion and Future Work

5.1 Conclusion

Reverse engineering binary network protocols poses significant challenges due to the presence of sub-byte field sizes and the complexity of information encoding schemes. Without publicly available specifications, analysts must rely solely on captured traffic; however, the compact nature of binary protocols often blurs the clear boundaries that are common in text-based designs. Unlike text-based protocols, which typically include separators or delimiters to indicate the start and end of fields, binary formats compress information to maximize efficiency. This compression makes it extremely difficult to identify where one field ends and the next begins, particularly in the presence of variable-length fields, optional payloads, and nested structures that further obscure the underlying format.

These difficulties have direct implications for practice. In critical security applications, such as deep packet inspection, anomaly detection, and threat intelligence, accurate protocol knowledge is essential to ensure correct traffic classification and to identify malicious activity. However, the lack of reliable field boundary detection in binary protocols undermines the precision of these applications, often resulting in incomplete or inaccurate interpretations. The problem becomes even more pronounced when field boundaries fall at arbitrary bit positions, where conventional byte-oriented methods fail to provide meaningful results.

Despite ongoing research efforts, no existing tools are capable of achieving accurate and comprehensive boundary detection for binary protocols. Prior work has made progress in handling certain protocol structures, but the fundamental challenges remain unresolved.

This situation calls for innovative approaches that can address the unique characteristics of binary protocols and support the increasing demands of modern network security analysis.

To address this gap, we developed Bit-to-Byte Alignment with Entropy (BBAE), a novel approach designed to effectively identify field boundaries of arbitrary length at the bit level. Traditional methods often struggle due to their reliance on rigid byte-aligned assumptions, which hinder their ability to recognize sub-byte fields and irregular structures commonly found in binary protocols. BBAE overcomes this limitation by employing a systematic strategy that integrates entropy analysis and bit-congruence calculations across multiple window sizes. Through this design, the method can capture and highlight positions that exhibit significant changes in randomness or similarity, thereby indicating a high likelihood of true field boundaries.

An essential component of BBAE is the padding mechanism. After identifying potential bit-level boundaries, BBAE pads sub-byte fields to achieve byte alignment without altering the underlying semantics. This step is crucial, as it allows the padded data to be processed using existing state-of-the-art byte-oriented protocol reverse engineering tools, such as BinaryInferno, which are highly effective when operating on byte-aligned structures. By doing so, BBAE bridges the gap between bit-oriented analysis and byte-oriented methods, enabling accurate validation of detected boundaries and the discovery of new ones that would otherwise remain hidden. In this way, BBAE not only establishes itself as the first powerful tool capable of addressing the structural complexity of binary protocol analysis, but it also introduces a promising method for augmenting existing byte-oriented analytical tools with the capability to handle bit-oriented analysis. This integration ensures that the strengths of both entropy-driven detection and established byte-oriented inference methods are combined, ultimately providing a more reliable and scalable solution for protocol reverse engineering in scenarios where field boundaries occur at arbitrary bit positions.

To facilitate the use of BBAE in practice, we further developed a Graphical User Interface (GUI) to assist the analysis and improve the overall usability of BBAE. The GUI provides an intuitive and interactive environment that automates intermediate operations, reduces repetitive manual tasks, and enables users to visualize and verify the inferred boundaries more efficiently.

5.2 Future Work

BBAE uses cross-verification to significantly reduce false positives by consolidating results from multiple window sizes and metrics. While this mechanism strengthens the reliability of detected boundaries and ensures more robust inference in diverse scenarios, it still faces several inherent limitations. First, it relies on data variability; when captured traces lack sufficient diversity, some low-variance fields, such as reserved bits or rarely triggered flags, may not be correctly identified. Second, the approach encounters challenges when dealing with encrypted or compressed network traffic, where entropy and similarity measures become less informative due to the near-uniform distribution of bit patterns. Third, the framework may face scalability issues when analyzing very large datasets or highly complex protocols, as the exhaustive evaluation of potential boundaries at multiple granularities increases computational cost. Finally, the current design assumes a big-endian format, which restricts its direct applicability to protocols using alternative encoding schemes. Our future work is to address the above limitations.

Bibliography

[1] Rahul G Andrew B. Mqtt version 3.1.1: Protocol specification, 2014. Available: `https://docs.oasis-open.org/mqtt/mqtt/v3.1.1/os/mqtt-v3.1.1-os.pdf`.

[2] Juan Caballero, Heng Yin, Zhenkai Liang, and Dawn Song. Polyglot: automatic extraction of protocol message format using dynamic binary analysis. In *Proceedings of the 14th ACM Conference on Computer and Communications Security*, CCS '07, page 317–329, New York, NY, USA, 2007. Association for Computing Machinery.

[3] J. Chandler, A. Wick, and K. Fisher. Binaryinferno: A semantic-driven approach to field inference for binary message formats. 2023.

[4] Weidong Cui, Jayanthkumar Kannan, and Helen J Wang. Discoverer: Automatic protocol reverse engineering from network traces. In *USENIX Security Symposium*, pages 1–14. Boston, MA, USA, 2007.

[5] J. Duchêne, C. Le Guernic, E. Alata, V. Nicomette, and M. Kaâniche. Protocol reverse engineering: Challenges and obfuscation. In *Risks and Security of Internet and Systems*, pages 139–144. Springer, 2017.

[6] W Eddy. Rfc 9293: Transmission control protocol (tcp), 2022.

[7] Yuyao Huang, Hui Shu, Fei Kang, and Yan Guang. Protocol reverse-engineering methods and tools: a survey. *Computer Communications*, 182:238–254, 2022.

[8] Zewen Huang, Kui Wu, Shengqiang Huang, Yang Zhou, and Ronnie Salvador Giagone. Automatic field extraction of extended tlv for binary protocol reverse engineering. In *2022 International Conference on Computer Communications and Networks (ICCCN)*, pages 1–10, 2022.

[9] Jiayi Jiang, Xiyuan Zhang, Chengcheng Wan, Haoyi Chen, Haiying Sun, and Ting Su. Binpre: Enhancing field inference in binary analysis based protocol reverse engineering. In *Proceedings of the 2024 on ACM SIGSAC Conference on Computer and Communications Security*, pages 3689–3703, 2024.

[10] Stephan Kleber, Henning Kopp, and Frank Kargl. Nemesys: Network message syntax reverse engineering by analysis of the intrinsic structure of individual messages. In *12th USENIX Workshop on Offensive Technologies (WOOT 18)*, 2018.

[11] Yuhuan Liu, Yulong Ding, Jie Jiang, Bin Xiao, and Shuang-Hua Yang. Industrial control protocol type inference using transformer and rule-based re-clustering. In *INFOCOM 2024*, pages 1–9. IEEE, 2024.

[12] Zhengxiong Luo, Kai Liang, Yanyang Zhao, Feifan Wu, Junze Yu, Heyuan Shi, and Yu Jiang. Dynpre: Protocol reverse engineering via dynamic inference. In *Proc. NDSS*, pages 1–18, 2024.

[13] Bowei Ning, Xuejun Zong, Kan He, and Lian Lian. Preiud: an industrial control protocols reverse engineering tool based on unsupervised learning and deep neural network methods. *Symmetry*, 15(3):706, 2023.

[14] Zhen Qin, Zeyu Yang, Yangyang Geng, Xin Che, Tianyi Wang, Hengye Zhu, Peng Cheng, and Jiming Chen. Reverse engineering industrial protocols driven by control fields. *INFOCOM 2024*, pages 1–10, 2024.

[15] C Shanon. A mathematical theory of communication. the bell systems technical journal, 27. *Mathematical Reviews (MathSciNet): 1948 MR10, 133e*, 1948.

[16] Siyu Tao, Hongyi Yu, and Qing Li. Bit-oriented format extraction approach for automatic binary protocol reverse engineering. *Iet Communications*, 10(6):709–716, 2016.

[17] Yapeng Ye, Zhuo Zhang, Fei Wang, Xiangyu Zhang, and Dongyan Xu. Netplier: Probabilistic network protocol reverse engineering from message traces. In *NDSS*, 2021.

[18] M. Zhan, Y. Li, B. Li, J. Zhang, C. Li, and W. Wang. Toward automated field semantics inference for binary protocol reverse engineering. *IEEE Transactions on Information Forensics and Security*, 19:764–776, 2024.

[19] Mengqi Zhan, Yang Li, Bo Li, Jinchao Zhang, Chuanrong Li, and Weiping Wang. Toward automated field semantics inference for binary protocol reverse engineering. *IEEE Transactions on Information Forensics and Security*, 19:764–776, 2024.

[20] Sen Zhao, Shouguo Yang, Zhen Wang, Yongji Liu, Hongsong Zhu, and Limin Sun. Crafting binary protocol reversing via deep learning with knowledge-driven augmentation. *IEEE/ACM Transactions on Networking*, pages 1–16, 2024.