NEMESYS: Network Message Syntax Reverse Engineering by Analysis of the Intrinsic Structure of Individual Messages

Stephan Kleber *stephan.kleber@uni-ulm.de*
Henning Kopp *henning.kopp@uni-ulm.de*
Frank Kargl *frank.kargl@uni-ulm.de*
Institute of Distributed Systems, Ulm University

Abstract

Protocol reverse engineering based on traffic traces allows to analyze observable network messages. Thereby, message formats of unknown protocols can be inferred. We present a novel method to infer structure from network messages of binary protocols. The method derives field boundaries from the distribution of value changes throughout individual messages. None of many previous approaches exploits features of structure which are contained within each single message. Our method exploits this intrinsic structure instead of comparing multiple messages with each other. We implement our approach in the tool NEMESYS: NEtwork Message SYntax analysiS. Additionally, we introduce the Format Match Score: the first quantitative measure of the quality of a message format inference. We apply the Format Match Score to NEMESYS and a previous approach and compare the results to mutually validate our new format inference method and the measure of its quality.

1 Introduction

A network protocol has the purpose to enable communication between nodes. Communicating nodes are entities implementing a specific shared protocol specification. In this context, a specification allows for the interpretation of transmitted data as information of defined meaning by the receiver. An analyst being confronted with communication which uses an unknown specification will need to reverse engineer its specification by monitoring network traffic or the communicating entities.

Traffic analysis is one method of protocol reverse engineering to infer the meaning of the communication [5]. It resorts solely to the analysis of traffic traces, which are observable on the link between the communicating entities. Although traffic analysis can only gain information from what can be observed on the communication link, it is non-invasive and does not require control over any entity; therefore, static traffic analysis is regularly applied. This method of network analysis has been used to gain comprehension of hitherto unknown network protocols for security-relevant tasks. Use-case examples for static traffic reverse engineering are the analysis of botnets [13], the setup of honeypots [13, 14], vulnerability testing by fuzzing [12], and the automated modeling of networks [21].

Current methods for traffic analysis are mainly based on algorithms originating from natural language processing and bio-informatics. Although these fields solve the related problem of inferring structure from sequences of values, in their context the challenge is reduced by additional knowledge about their data. Natural language processing determines words from delimiters, separating atomic parts of the vocabulary, and then identifies keywords by their frequency of occurrence. In bio-informatics it is often known which sequences of amino-acids are relevant in DNA or protein-sequences, therefore, aligning multiple sequences on their sub-sequences is a robust method of identifying similarity between the sequences.

Network protocols do not necessarily provide the required properties to apply either of these methods efficiently: A lot of protocols do not use keywords that would be discernible by natural language processing; in bio-informatics only few different sequences need to be aligned at a time. In contrast, inferring message formats from network traffic traces requires a large number of messages of one protocol to observe the variability of values. However, large corpora of message traces critically impact the performance, due to the exponential complexity [18] of multiple sequence alignment. Therefore, neither natural language processing nor bio-informatics can provide for sufficient inference of the structure of binary data in a network protocol message.

Subsequent analysis steps of protocol reverse engineering, like message type identification, semantic deduction, or behavior analysis to derive the state model of

the protocol, are highly dependent on the format inference quality. To cluster message types or align field sequences more precisely, the main goal of message format inference is to reliably and automatically segment each message on byte-positions that constitute field boundaries.

Contributions: Our paper contains two major contributions: Foremost, we propose a new method to infer structure from network messages. Additionally, we introduce the first quantitative measure of the quality of a message format inference.

To infer structure from network messages, we exploit typical patterns of value changes in network messages as heuristic for field boundaries. This yields a message segmentation derived from the distribution of value characteristics throughout each individual message. The basic idea of this method is to exploit the intrinsic structure of each single message for the analysis, instead of comparing multiple messages with each other. This approach obviated the need for pairwise comparisons of messages as the initial feature extraction step of protocol reverse engineering. We implement our approach in the tool NEMESYS: NETWORK MESSAGE SYNTAX ANALYSIS.

To evaluate NEMESYS and to compare multiple inference results, we introduce the measure Format Match Score (FMS), which quantifies the quality of the format inference for a specific message. We apply this measure to our results and compare them to the sequence-alignment field inference performed by the tool Netzob[1].

This article is structured as follows: In the next section we discuss related work and revisit the relevant common terminology for the discussion about network messages. In Section 3 we present the details of our approach. In Section 4 we describe the design of its implementation NEMESYS. We define the Format Match Score, our quality measure for network message syntax inference, in Section 5. Section 6 contains the results of our evaluation of NEMESYS by applying FMS. Finally, we outline our ideas for future work and conclude the paper in Sections 7 and 8.

2 Related Work

Static traffic analysis is a specific kind of protocol reverse engineering, where network traffic between two genuine entities is monitored purely passively. First solutions have been proposed by Beddoe [1] and Rauch [16]. The basic application of sequence alignment to network protocols originates from Beddoe. Based on his work, the most versatile available tool implementing static traffic analysis is Netzob [4]. We utilize Netzob's functionality to leverage our approach.

Since Beddoe's paper, algorithms from natural language processing and bio-informatics have been applied to network protocols.

Methods based on natural language processing work well on protocols which use ASCII-encoded keywords to structure their messages. Figure 1 shows part of a textual protocol with separator chars clearly delimiting keyword and data fields. Binary protocols, packing data more densely and not separating fields by delimiters or labeling them by explicit keywords, do not exhibit these characteristics, which, however, are necessary for natural language processing. Therefore, these methods inherently are not applicable to binary protocols.

As for bio-informatics algorithms, to derive a potential structure from an alignment of multiple messages, it requires that the same value is transmitted in the same relative position of a sequence of multiple values within each message. However, it cannot be unconditionally expected to find the same value in the byte sequence of multiple independent messages in binary protocols. On the other hand, long variable message parts bias the alignment by spurious relationships of values across messages and therefore lead to false positive identification of field boundaries.

ScriptGen [14], Discoverer [8], and Netzob use sequence alignment to infer message formats. Both, ScriptGen and Discoverer differ from Netzob in that they align subsequences of messages (tokens) instead of single bytes. They propose effective methods to generate such tokens from textual protocols. In addition, ScriptGen proposes to derive tokens from frequency, variance, and other byte characteristics throughout all messages of a trace. Although ScriptGen and Discoverer envision their methods to be universally applicable, they leave it to future work to solve the details of inferring binary protocols.

ProDecoder [19] and PRISMA [13] use statistical methods known from natural language processing. Both tools use these methods to find tokens by identifying sequences that appear together frequently and coherently. Due to the missing keywords and separators in binary protocols, natural language processing produces very limited results and is inefficient for large traces.

FieldHunter [3] is a recent promising approach combining concepts from Netzob, Discoverer, ScriptGen, and other related methods. It provides advanced solu-

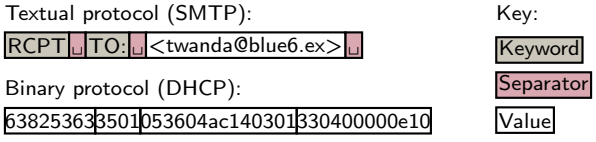


Figure 1: Snippets of textual and binary protocols.

tions for a number of challenges for format inference, like characterization of field types. However, FieldHunter does not exploit features contained inside one single message. Thus, like previous methods, it misses a lot of details of the structure of messages.

Despite this wide variety of methods and features which have been proposed for traffic analysis, to the best of our knowledge, a feature extraction from the intrinsic structure of messages has never been proposed in this context. Moreover, we are not aware of any kind of measure for the correctness and precision of the format inference of network messages. Therefore, we propose the first such measure in this paper.

The surveys by Duchêne et al. [9] and Narayan et al. [15] provide an overview of protocol reverse engineering beyond the methods directly related to our work, in particular such that are based on software reverse engineering. Software reverse engineering for the analysis of an executable that implements a network protocol has been proposed and is well understood for scenarios where this approach is applicable [6, 20]. In contrast to inferring the parsing logic of an executable program, traffic analysis remains possible in cases where the executable program binary is not accessible for reverse engineering. We do not further discuss these methods, since they utilize fundamentally different concepts to perform their task.

We adopt the **terminology** for protocol elements of Narayan et al. [15]: For the message type to be discernible and for the information to be interpretable at the receiver, each message needs a defined format following a strict syntax. This syntax defines fields, and the specification associates them with semantic. Each field has a length and a value. The length can be fixed or variable. The value can be completely variable, like in a data field, or it can be static, like a message type identifier, or it can be ephemeral. Ephemeral values remain static during exactly one session, such as the Identification field in DNS, intended to match request and reply despite using the connectionless UDP transport.

3 Approach

To discern the syntax of a network message, we utilize the structure that is intrinsic to the message. Since each network protocol is designed to be efficiently parseable by its recipient, we can expect to find specific hints towards the structure in each message itself. For example, binary protocols typically use fields of the lengths of common data types, such as 32 bits for an integer. However, field contents do not uniformly fill the value domain of such a fixed numerical field.

For instance, numeric values are a common field data type in network messages. Counted numbers exhibit a specific variance distribution from its most to its least significant byte. This observation is similar or even correlated to Benford's Law [2] which predicts the anomalous distribution of the digits of numbers found in the real-world. To illustrate, this is most obvious in the case of multiple zeros being the most significant bytes of a fixed size integer. For example, a 4 byte integer field containing the value 2069 would look like `00000815` in hexadecimals.

The observation of network messages shows a typical behavior throughout the sequence of values, which is indicating substructures of messages. Our approach is to segment a message according to features of its intrinsic structure. This way we determine field candidates. After investigating a number of different methods to characterize bit sequences [7], we discovered the similarity of consecutive message bytes to be a good feature to perceive the intrinsic structure of a message.

3.1 Similarity Revealing Structure

The main similarity feature we use is the **delta of the congruence in bit values** of consecutive bytes (BCD) of one message. We assume that the message does consist of multiple bytes, though our exposition is generalizable to work with nibbles or words instead. We define Bit Congruence as the bitwise similarity of bytes and utilize the bitwise similarity defined by Sokal and Michener [17]. We apply the Bit Congruence per each two consecutive bytes of the message.

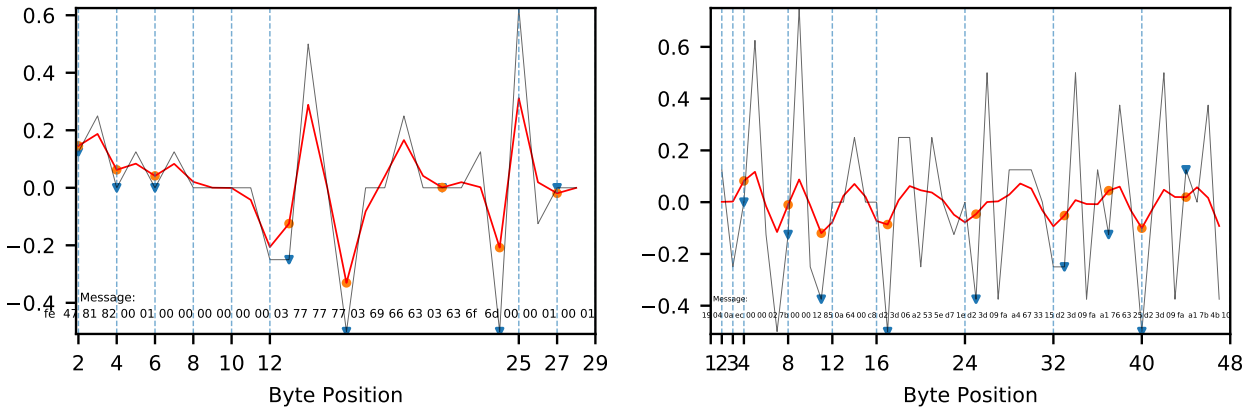
For two bytes b and $\overline{b}$ their bits are denoted as b^i and $\overline{b}^i$, with $0 \leq i < 8$. The number of bits that have the same value is called $c_{\text{agree}}(b,\overline{b}) = |\{0 \leq i < 8 : b^i = \overline{b}^i\}|$. The similarity metric by Sokal and Michener applied to b and $\overline{b}$ yields what we call the Bit Congruence

$$\text{BC}(b,\overline{b}) = \frac{c_{\text{agree}}(b,\overline{b})}{8}.$$

Iterating over all bytes $m_0,\ldots,m_n$ of a network message m, we can determine a delta between the Bit Congruences of consecutive byte pairs. It is calculated from the difference of the Bit Congruence in every pair of consecutive byte positions k. The delta of this similarity within a message of m bytes length then is the following vector.

$$\Delta\text{BC} = \big(\text{BC}(m_k,m_{k+1}) - \text{BC}(m_{k-1},m_k)\big)_{0<k<n}$$

Although bit values of longer sequences of numerical data tend to have a recognizable BC pattern in network messages, not all subsequences of bits obey this behavior individually. Thus, the BC is a noisy feature and consequently also ΔBC. Despite the noise, we need to reliably find positions in the message where the feature property changes noticeably across multiple bytes to discover the



(a) Features of one DNS message

(b) Features of one NTP message.

Figure 2: Gaussian filtered Bit Congruence deltas as message feature: The red line is showing the smoothed ΔBC; the fine black line ΔBC without smoothing; vertical blue dashed lines are true field boundaries; the byte values of the displayed message are shown along the x-axis in hexadecimals.

overall tendency. Therefore, we smooth the ΔBC by a standard Gaussian filter $g_\sigma(\Delta\mathrm{BC})$ with parameter σ.

Even by simple observation in Figure 2, this smoothed Bit Congruence delta clearly shows a periodic behavior coinciding with fields as postulated.

Numbers have a characteristic distribution of Bit Congruence deltas across their binary representation in one field. In particular, at the transition from one field to another, big-endian numeric values commonly have low variability; from the beginning of the field, this rises into high variability towards the end of the field. For counting numbers, this is due to the order of the significance of the bytes. We acknowledge that there are several data types of fields used in binary protocols that do not change their variability in a way recognizable by our feature. Some examples are char sequences (byte positions 13 to 25 in Figure 2a), seemingly random session cookies, message authentication codes, and nonces. While consecutive random-looking values cannot be split into their components by any method without additional specification, char sequences may change their variability according to the letter frequency in natural language. This is difficult to detect with our method, which compares only small subsequences of the message. However, seemingly random values occur comparatively seldom throughout a message and are therefore clearly discernible by our feature from other parts of the message. Character strings on the other hand are easily identifiable by their value domain, as we will show in our implementation (Section 4).

For an expedient analysis during steps after the extraction of the ΔBC feature, a large number of messages is needed to have a diverse set of samples. Only then the different realizations of potentially multiple syntaxes which are specified in one protocol become observable. The reason we chose Sokal and Michener to provide the Bit Congruence feature for our approach is twofold. For one, it is simple to be calculated so that we can efficiently analyze the necessary large numbers of independent messages. On the other hand, it is a normalized bit similarity metric. The normalization by the length of the compared bit strings has one conceptual and one practical benefit: It makes the approach universally valid, independently of the applied basic unit size. For common protocols, these basic units are bytes, but nibbles or larger bit strings may be used alternatively. The practical benefit of normalizing is that it allows to apply optimized default Gaussian-filter implementations, based on the normalized Gaussian distribution. These implementations require the sequence for smoothing to be normalized to values between zero and one.

3.2 Applying the Feature

We interpret the ΔBC feature to derive message segments that constitute field candidates for each analyzed message. Due to the explained typical feature-value distribution throughout one field, the segmentation is achieved by determining approximates for the **inflection points** at the **rising edges** of the discrete **Gaussian-filtered Bit Congruence delta** $g_\sigma(\Delta\mathrm{BC})$. By reducing the noise in the input data, the smoothing by the Gaussian filter shows the overall variability level for longer sequences of bytes in the message. It thereby prevents longer field candidates to be split up erroneously for a single deviating byte-value outlier. Figure 2 shows plots of unsmoothed and Gaussian-filtered ΔBC of two example messages. The byte position next to an inflection point on the rising

edge of the smoothed ΔBC is marked with an orange dot. The true field boundaries, we seek to infer, are marked by vertical dashed blue lines.

The number of discrete values to deduct inflection points from often are derived from only 2 to 4 bytes. Therefore, we do not use interpolated inflection points of $g_\sigma(\Delta\text{BC})$ directly. The exact cut position for segments at the rising edge of Bit Congruence deltas is determined from the maximum delta of the unsmoothed ΔBC in scope of each rising edge. Therefore, we determine the scope of a rising edge from the interval of a pair of local extrema $(e_{\min}, e_{\max})$. As cutting point, we use the most distinctive change in each interval (e_n, e_{n+1}) for each n where e_n is a local minimum and e_{n+1} is a local maximum. This yields exact points of transitions from high to low variability throughout one message. To illustrate, our approximation of the inflection points is marked by blue triangles in Figure 2. Due to the noisy nature of the feature, the resulting message segments are not necessarily cut at the exact field boundaries and may slightly skitter compared to the protocol specification.

Our approach now provides segmentation of messages by approximating the inflection points of Gaussian-smoothed deltas of Bit Congruences ($g_\sigma(\Delta\text{BC})$).

4 NEMESYS Implementation

We implement our approach in the tool NEMESYS: NETWORK MESSAGE SYNTAX ANALYSIS.

4.1 Architecture

NEMESYS is implemented in Python 3 as a proof-of-concept and consists of the modules SpecimenLoader, MessageAnalyzer, and MessageSegment depicted in Figure 3.

The SpecimenLoader reads a PCAP file and prepares its representation for the analysis using methods of Netzob. MessageAnalyzer defines classes for each analysis method, as well as an AbstractBaseClass for all of them. Bit Congruence (BC), Bit Congruence delta (BCD), and Gaussian-filtered BCD (BCDG) are implemented as analyzer subclasses. The BCDG class also contains methods to approximate the inflection points and to cut the message into according MessageSegments.

4.2 Segmentation

In NEMESYS, a MessageSegment is fully defined by the message it originates from, the byte offset, the length in bytes, and the kind of feature analysis performed on the message to gain the segmentation. This encapsulates a message together with an analysis method and allows to slice the message according to the extracted features.

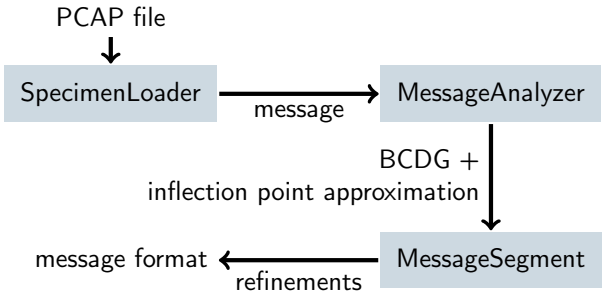


Figure 3: Overview of the NEMESYS analysis process

Features are one or more analysis values that are used as basis to detect a field boundary. Moreover, the architecture to calculate and interpret features enables to flexibly exchange the analysis method by writing a new analysis-class that can be instantiated in the message segments.

The sole parameter of NEMESYS is the radius of the smoothing by the Gaussian filter denoted as σ. σ is the standard deviation of the Gaussian distribution that is used as the kernel to convolute with the BCD value. To illustrate the influence σ has on the smoothing, regard another application of the Gaussian filter: image processing. In this context, Gaussian filtering blurs the image to reduce noise such as scratches and raster artifacts. It produces a weighted mean of each data point and its neighbors.

In our application to network messages, the optimal value for σ is dependent on the field lengths of the protocol. This parameter can be adjusted by the analyst to improve the message segmentation precision. However, for unknown protocols a reasonable assumption about the typical field length must be made. Our empirical tests have shown that for common protocols with field lengths of 2 to 8 bytes, a value of 0.9 yields best field matching results (see Appendix, Table 4 and Figure 10).

4.3 Refinements

It is possible to make computationally cheap refinements of the results. These are intended to take additional knowledge about typical protocol design into account. We implemented one kind of value-based refinement to demonstrate the procedure. In particular, we address the issue of character sequences, which do not exhibit the same structure as numerical fields. Such fields may very well be embedded within binary protocol messages, like DNS containing a domain name.

Typically, character sequences are split into segments too short. In some cases, segments before or after character sequences contain one or more character bytes. Therefore, NEMESYS merges consecutive segments which completely consist of printable-character

values into one text field. We chose to define printable characters according to the ASCII encoding as either \t, \n, or \r, or having a value between $\geq$ 0x20 and $\leq$ 0x7e. Afterwards, NEMESYS checks whether segments adjacent to text segments need to be split and re-segmented to keep consecutive chars together. These operations have almost no effect on purely binary protocols but improve the inference of text fields within binary protocols.

5 Format Match Score

We introduce the Format Match Score (FMS) as quantitative measure that provides a reference value for the quality of a format inference. To the best of our knowledge, this score is the first of its kind to adequately incorporate the relevant aspects of correctness of a message format inference. It is designed to (1) take into account the ratio of correctly recognized fields, (2) differentiate between shifted field boundaries and completely wrongly inferred fields, which result in too many or too few fields of one message, and (3) quantify the decreasing utility of the different aspects of deviation of the inference. Deviations and ratios are determined by applying the protocol specification to a specific inferred message. The FMS is intended as a benchmark for the message format inference quality to be applied with a known protocol as representative test specimen. It cannot serve as an oracle for the quality of the inference of a truly unknown protocol. Nevertheless, knowing the efficacy of a format inference algorithm by benchmarking it allows to deduce the expected result quality of the approach for an unknown protocol of similar kind.

5.1 Quality Aspects

The FMS is based on a number of quality aspects as introduced above. To formalize these, we first define the necessary properties of a message, its real format, and its inference.

For a single message, we define the properties:

R: The set of real field boundaries and $|R|$ their number.

I: The set of inferred boundaries and $|I|$ their number.

r_k with $0 < k < |R|$: The byte index of the kth real field boundary; to ease notation, we always set $r_0 = 0$ at the start of the message and $r_{|R|}$ as the message length.

i_l with $0 < l < |I|$: The byte index of the lth inferred field boundary.

We define a scope for each true field boundary r_k within a single message. A scope begins at the center between the previous and the current boundary r_k and ends right before the center between the current and the subsequent boundary as illustrated in Figure 4. The message start r_0 and end $r_{|R|}$ have no scopes assigned and are not considered inferable boundaries. Thus, an inferred boundary i_l is within the scope of r_k for any $0 < k < |R|$ if the following inequality holds:

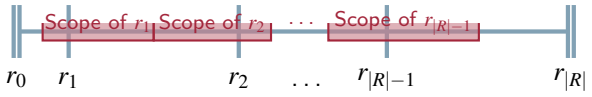


Figure 4: Scope of each true field boundary.

$$r_{k-1} + \frac{r_k - r_{k-1}}{2} \leq i_l < r_k + \frac{r_{k+1} - r_k}{2} \qquad (1)$$

We define $\boldsymbol{\delta_r}$ as the **distance** of a real field boundary r_k to the nearest inferred field boundary i_l:

$$\delta_r = \min\{|i - r|, \text{ where } i \text{ satisfies Equation (1)}\}$$

As usual, we define the minimum operator on an empty set to be $\min \emptyset = -\infty$. Consequently, δ_r is interpreted as:

$\delta_r = -\infty$**:** *no* matching inferred boundary exists for the true boundary r

$\delta_r = 0$**:** *an exactly* matching inferred boundary exists for the true boundary r

$-\infty < \delta_r < 0$**:** *the nearest* inferred boundary for the true boundary r is δ_r bytes left of r

$\delta_r > 0$**:** *the nearest* inferred boundary for the true boundary r is δ_r bytes right of r.

5.2 Calculating the Format Match Score

Knowing R, I, and δ_r, we can now calculate the FMS.

We use Gaussian weights to assign a measure of quality $\exp\left(-(\delta_r/\gamma)^2\right)$ to an inferred field with field distance δ_r. The parameter γ decreases the steepness of the quality drop towards larger deviations from zero of the distance between inferred and real field boundaries. Depending on the use case, γ can be adjusted for benchmarks to represent the requirements of the use case in regard to the accuracy of the format inference.

The Format Match Score for one message then is

$$\text{FMS} = \underbrace{\exp\left(-\left(\frac{|R| - |I|}{|R|}\right)^2\right)}_{\text{Specificity penalty}} \cdot \underbrace{\frac{1}{|R|}\sum_{r \in R} \exp\left(-\left(\frac{\delta_r}{\gamma}\right)^2\right)}_{\text{Match gain}}$$

The **specificity penalty** incorporates the deviation of the inferred field count from the true amount of fields in the format. Any deviation from zero reduces the

δ_r	$\exp\left(-\left(\frac{\delta_r}{\gamma}\right)^2\right)$	Note
0	1	exact match
± 1	0.779	
± 2	0.368	
± 3	0.105	
± 4	0.018	
$-\infty$	0	no matching inferred field

Table 1: Rounded weights of inferred fields corresponding to real fields r at distance δ_r for $\gamma = 2$.

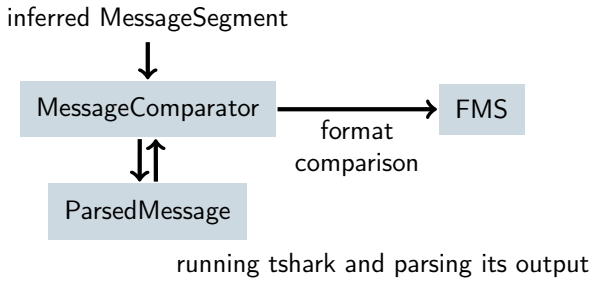


Figure 5: Overview of the evaluation process

FMS proportional to a Gaussian distribution. The non-linearity of the penalty takes into account that the usefulness of the inference typically decreases slowly for only few additional or missing fields, but the inference becomes useless with increasing pace for larger skitter.

The **match gain** incorporates near field matches, which are weighted by the distance from their true field boundary positions, and exact field matches. It is normalized by the amount of true fields in the message format. The weight of an exact match is 1 and of a true field that lacks any counterpart in the inference it is 0. Each near match is weighted non-linearly by a Gaussian distribution dependent on the distance of the true from the inferred field. Table 1 gives some examples for values of near matches, assuming a factor of $\gamma = 2$.

The overall FMS assumes 1 for an exact match of the inferred and true format. It approaches 0 for increased deviation of inferred and true format.

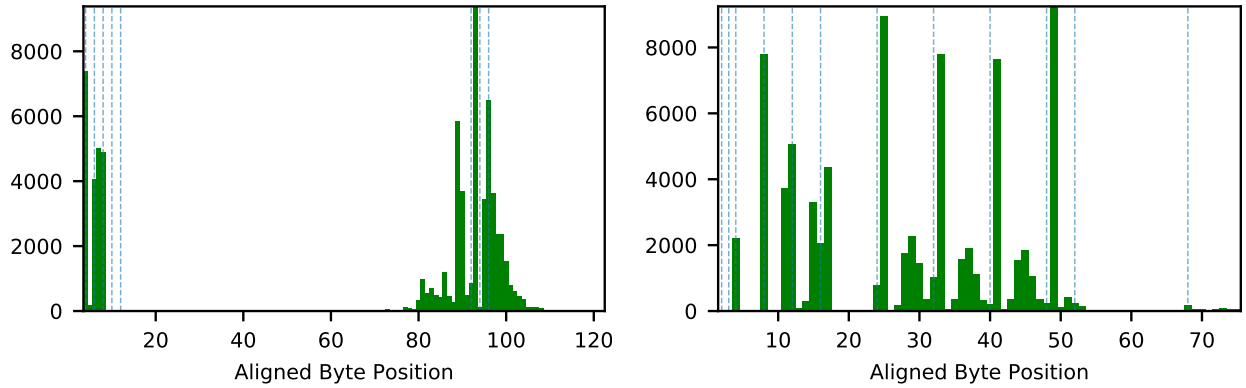
6 Evaluation

We evaluate the quality of our format inference approach by applying the FMS to the inference results of our method's implementation NEMESYS. For the evaluation, we implemented the Python-modules MessageComparator and ParsedMessage as add-on of NEMESYS. The two modules obtain the dissection of each message and compare these real field boundaries to the inferred ones by applying FMS. We illustrate this process in Figure 5.

As baseline information about the protocol specification, we utilize tshark[2]. For each message in the trace, we compare the inference results to the according protocol dissector provided by tshark. ParsedMessage hands each message to a tshark process and parses the JSON output of its dissectors. As specimens, we use the binary protocols DNS, NTP, and DHCP. We chose these protocols as representatives of different typical binary protocols. DHCP has varying amounts and lengths of fields. We chose NTP because it is a protocol of fixed field lengths, where the lengths range from 1 to 8 bytes for the different fields. DNS contains mostly 2 byte binary fields mixed with variable length fields of ASCII-encoded characters. The traces we analyzed are publicly available[3]; we preprocessed each raw trace by removing duplicates of the payload and truncating them to traces of the sizes of 100, 1 000, and 10 000 messages.

To visualize the accuracy of the inference for a whole trace, we generated histograms of the amounts of inferred boundaries positioned around true boundaries of fields. Figure 6 depicts this histogram for 10 000 messages of DNS and NTP each. (We placed the respective plot for DHCP in the Appendix, Figure 8.) In these plots, the vertical dashed blue lines denote true boundaries; the bars count in how many messages a boundary was inferred at that byte position. Since our format inference method is heuristic, inferred field boundaries are scattered around the true field boundaries. The narrower the bars of the plot are scattered around the true fields, the better the inference matches the true format. To determine the distance between each true field boundary and its nearest inferred counterparts, we needed to take the variable length fields of DNS and DHCP into account. Therefore, we aligned the true field boundaries and their nearest inferred counterparts across all messages for the graphical representation to the maximum length of each field in the messages' sequences.

To provide a quantitative evaluation, we applied our novel FMS. We chose the parameter of the FMS to be $\gamma = 2$ in our evaluation so that small deviations in the inference still are rewarded as shown in Table 1. Thus, a distance of an inferred field by 1 still is rewarded by about 0.78 of an exact match, whereas for a distance of 4 byte positions only less than 0.02 remains. For comparison with our results and to validate the FMS, we used Netzob to infer the same messages as NEMESYS. Table 2 shows the inference results in FMS for the best matching message in the trace and the average inference quality over all messages in the trace. We repeated the analysis with traces of different sizes and recorded the runtime of each analysis.



(a) Inferred boundary positions for 10 000 DNS messages.

(b) Inferred boundary positions for 10 000 NTP messages.

Figure 6: Histograms of inferred field boundaries around true field boundaries. The vertical dashed blue lines denote true boundaries; the bars count in how many messages a boundary was inferred at that byte position.

		Netzob		NEMESYS		
	Trace size	100	1 000	100	1 000	10 000
FMS (best)	DNS	0.56	0.56	0.60	0.68	0.71
	DHCP	0.53	—[4]	0.66	0.70	0.73
	NTP	0.71	0.66	0.61	0.66	0.67
FMS (avg.)	DNS	0.56	0.54	0.47	0.45	0.45
	DHCP	0.44	—[4]	0.52	0.53	0.54
	NTP	0.56	0.39	0.47	0.44	0.45
runtime in sec.	DNS	0.6	246	0.1	1.0	10
	DHCP	15	2224	0.5	4.3	47
	NTP	0.7	312	0.1	0.9	10

Table 2: Format inference results for best and average quality of all messages measured in FMS.

Both, Netzob and NEMESYS require a parameter, which we needed to set for our test runs. To select these, we iterated the respective parameters during test runs of each tool. We then used the best NEMESYS σs and Netzob similarity-thresholds for each protocol, according to the FMS. For Netzob the similarity thresholds used were 53 (DNS), 75 (DHCP), and 66 (NTP); for NEMESYS we used $\sigma = 0.6$ (DNS, DHCP) and 1.2 (NTP).

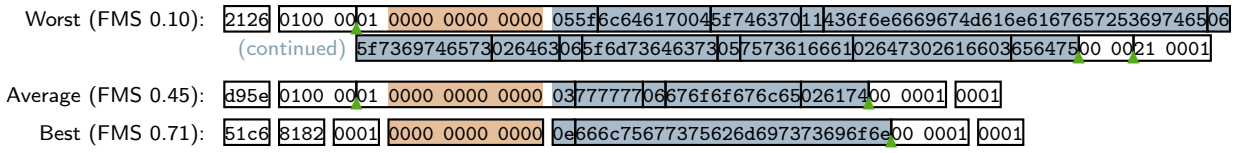
In the best case inferences, NEMESYS shows moderate (DNS) to significant (DHCP) better format matching than Netzob, while both approaches yield comparable results for NTP. In the average case, Netzob and NEMESYS yield a similar FMS, partly with Netzob leading (DNS, 100 NTP messages), partly with NEMESYS leading (more than 100 NTP messages). Overall, we conclude that the inference quality of Netzob and NEMESYS is similar.

What sets NEMESYS apart is that it does not need to do any comparison between multiple messages in the trace. Dependent on the length and number of messages, this results in a linear complexity of the analysis. On the other hand, global multiple sequence alignment — performed by Netzob and most other protocol reverse engineering tools — has exponential runtime. It guarantees to optimally align k sequences of the maximum length l at the complexity of $\mathcal{O}(l^k)$ [10]. The analysis run of 10 000 messages could not be performed with Netzob due to the exponential increase in its runtime and memory consumption. The runtimes of the analyses in Table 2 show this advantage of our approach in scalability. This becomes even more distinct when considering the maximum lengths of the messages in our the analysis traces: NTP and the DNS trace we used contain small messages of 68 and 96 bytes at max. Our DHCP trace contains largest messages of 548 bytes. Neither of these lengths are unusual and should not be prohibitive for an automated message analysis. NEMESYS completes the segmentation of even the largest trace of longest messages in under one minute, while Netzob requires almost 40 minutes for only the tenth of this amount of messages, as can be seen regarding DHCP in Table 2.

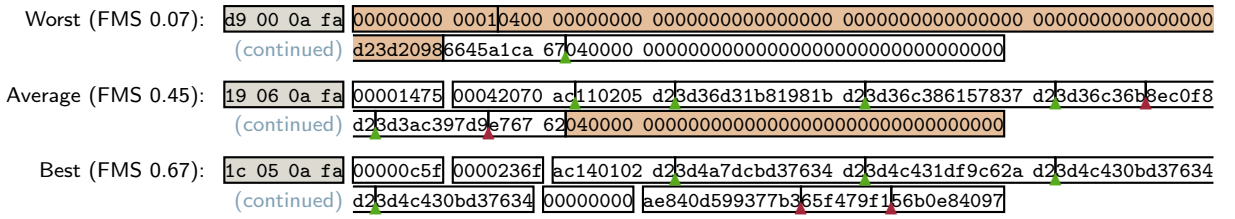
6.1 Limitations

A general limitation of static traffic analysis is that it is prevented by encryption of the messages. This can only be overcome by obtaining a plain-text trace. One method to accomplish this is memory introspection during the runtime of a program before or after encryption [9]. Therefore, this approach is not a method of static traffic analysis. Alternatively, a Man-in-the-Middle between two genuine entities can record the decrypted messages in transit [11]. Both methods require control over either the entities' execution environment or over the network topology.

Due to the nature of our approach, being a heuristic method, there are some limitations regarding the result



(a) Inferred DNS messages (worst, average, and best case)



(b) Inferred NTP messages (worst, average, and best case)

Figure 7: Example segmentations of messages from the specimen traces annotated with the comparison to the dissector: True fields are separated by SPACEs; inferred fields are in framed boxes. Values are in hexadecimal notation.

		FMS	I	M	N	S	Match Gain	Specificity Penalty
DNS	worst	0.10	17	1	4	-9	0.37	0.28
	average	0.45	8	2	3	0	0.45	1.00
	best	0.71	7	5	1	1	0.72	0.98
DHCP	worst	0.20	6	4	2	12	0.31	0.64
	average	0.54	21	7	9	2	0.54	0.992
	best	0.73	25	13	7	-1	0.73	0.998
NTP	worst	0.07	4	1	2	8	0.12	0.64
	average	0.45	10	2	6	2	0.47	0.97
	best	0.67	11	5	4	1	0.68	0.99

Table 3: Quality aspects examples (rounded) from the evaluation analysis runs of NEMESYS with 10 000 messages. M is the number of exact matches, N of near matches. For the sample messages see Figures 7 and 9.

precision. To illustrate what kind of errors account for a suboptimal inference by NEMESYS, we discuss example message segmentations annotated with true fields for comparison. For each protocol, we present the best, average, and worst case FMS of an inferred message format in Figure 7. (Due to the message length, we placed an abbreviated depiction of DHCP messages in the Appendix in Figure 9.)

Besides other quality aspects, Table 3 gives the number of over- and underspecified field boundaries in column S. The specificity $S = -9$ for the message of **DNS** with the **worst** inference result may be read as NEMESYS having inferred nine fields too much. Figure 7a shows that this is due to the character sequence embedded in the message blue. Three fields have not been inferred due to unset values orange. They contain only zero-bytes with no indication of field boundaries in between. This sums up to the nine overspecified fields. The three near field matches are off-by-one errors due to the heuristic feature (green ▲). In the **average** case of DNS inferences, the character sequence leads to three fields too much. The same amount of fields is missing due to unset fields with zero-values . In the **best** case, two such fields are not inferred. The near field match (green ▲) is due to the heuristic.

Regarding **NTP** in Figure 7b also shows unset fields with zero-values causing a bias at the previous and following field. For all shown NTP messages, the heuristic causes an off-by-one error at several field starts (green ▲). Two positions in the average and best cases are misinterpreted as boundaries (red ▲). An example is the last field of the best NTP match. This is a message authentication code, and as such a pseudo-random value not a counting number, to which neither Benford' Law nor any similar principle applies. Moreover, our method cannot infer single byte fields, like the four values at the beginning of the NTP messages beige.

While our evaluation shows a good approximation for areas of potential fields, pinpointing of the exact field boundaries is challenging with this feature. Despite some mismatches due to the heuristic approach, our evaluation shows that the inflection points in our feature of Bit Congruence deltas yields reasonably precise results. Furthermore, by abstracting from concrete binary values to a heuristic feature, we are able to discover structural patterns which remain hidden when only exact byte value matches are considered. Bearing in mind that we only analyze one message at a time, NEMESYS is able to infer message formats with competitive precision and excellent performance.

7 Future Work

NEMESYS generates message segments from the change in Bit Congruence between bytes. Besides this similarity metric, a large number of other metrics exist [7] that could be evaluated to extract an alternative binary similarity feature that reveals message structure. For example, a similarity which weights agreement on 1-bits over agreement on 0-bits may be more suited to discriminate flag-like fields.

NEMESYS' message segments are heuristic and therefore fuzzy in regard to the inferred field boundary. To pinpoint the exact boundaries, specific characteristics of the Bit Congruence or value based refinements in the vicinity of field candidates may be used. Learning characteristics of these features from fields in known protocols could also help to recognize the data type and decrease the skitter for non-numeric field types.

For an overall quality value, we used the average FMS over the messages of a trace in the evaluation. Adding single messages that contain more inferable structure increases the overall quality score of the larger trace. Inversely, eliminating the worst-inferred messages effectively increases the quality. This is easy for a known protocol, for which a FMS can be calculated; for an unknown protocol it is impossible. Thus, it remains to be solved how to identify messages that do not exhibit enough discernible structure for syntax analysis.

As the next step in the protocol reverse engineering process, the analyzed messages need to be classified into clusters of equal format. Thus, the resulting field candidates can be combined into field "templates" according to their similarity in terms of the Bit Congruence or other features. As described above, these features can be used to hypothesize about the data type of the field. Combining the templates with their hypothetical field type renders it possible to utilize a method like ScriptGen's region analysis [14] or Discoverer's type-based alignment [8] Such kind of type-based alignment and clustering can use this information to more efficiently categorize messages without having to align byte-by-byte. To evaluate methods using data type identification, the FMS could be enhanced to assess the inference of field data types.

For use cases like anomaly detection, it may not be required to exactly know the format but a characterization of messages to recognize their protocol without knowing the exact specification. Our method could be adapted not to segment messages but to characterize them in whole by Bit Congruence features. Such a characterization could be applied as fingerprint of allowed protocols without requiring deep packet inspection.

8 Conclusion

In this paper, we present a novel method of format inference for unknown network protocol messages. Moreover, we introduce a measure to quantitatively compare the quality of format inferences.

NEMESYS segments messages without relying on identical byte values to determine similarity. The novelty of our approach is that we do not compare multiple messages to find similarities in byte values, but that we analyze a single message at a time to discover its intrinsic structure. With this method we are able to efficiently identify the message syntax of binary protocols. The resulting message segments need further interpretation to determine the exact field boundaries reliably. Nevertheless, we achieve results comparable to sequence alignment by analyzing only isolated messages. Compared to the exponential complexity of any pairwise comparison, we reveal structure more efficiently, i. e., scaling with the message amount and the message sizes in linear time. Moreover, by abstracting from concrete binary values to a heuristic feature, we are able to discover structural patterns, which remain hidden when only exact value matches are considered.

Not having enough information about a protocol specification makes guessing the quality of a reverse engineering result a vague task. Evaluating a format inference method, like Netzob or NEMESYS, using known protocols allows a quantitative comparison. The Format Match Score we introduce is the first of its kind to measure the quality assessment of a method. It takes into account the specific measurable aspects of the divergence between inferred and true message formats. Our evaluation applies the Format Match Score to NEMESYS, mutually validating both contributions of this paper.

The specific benefit of the approach of NEMESYS is that it works without having to analyze arbitrary parts of all messages in a set of specimens. With further refinement, it provides the means to (1) pinpoint the exact field boundaries from the heuristic candidates it yields, (2) identify field types through the feature characteristics, and (3) identify message types from the feature profile.

Notes

[1] `http://netzob.org/`

[2] `https://www.wireshark.org/docs/wsug_html_chunked/AppToolstshark.html`

[3] NTP and DHCP filtered from `http://download.netresec.com/pcap/smia-2011/`; DNS filtered from `http://ictf.cs.ucsb.edu/ictfdata/2010/dumps/ictf2010pcap.tar.gz`

[4] We considered Netzob's parsing of the already aligned 1 000 DHCP messages as failed after 100 hours of runtime.

All URLs last accessed on May, 20 2018.

References

[1] Marshall A. Beddoe. *Network Protocol Analysis using Bioinformatics Algorithms*. Tech. rep. McAfee Inc., 2004.

[2] Frank Benford. “The Law of Anomalous Numbers”. In: *Proceedings of the American Philosophical Society* 78.4 (1938), pp. 551–572.

[3] Ignacio Bermudez, Alok Tongaonkar, Marios Iliofotou, Marco Mellia, and Maurizio M. Munafò. “Towards Automatic Protocol Field Inference”. In: *Computer Communications* 84 (June 2016).

[4] Georges Bossert, Frédéric Guihéry, and Guillaume Hiet. “Towards Automated Protocol Reverse Engineering Using Semantic Information”. In: *Proceedings of the 9th ACM Symposium on Information, Computer and Communications Security*. ASIA CCS ’14. ACM, 2014.

[5] CAPEC Content Team. *CAPEC - CAPEC-192: Protocol Reverse Engineering (Version 2.6)*. June 2014.

[6] Chia Yuan Cho, Domagoj Babić, Pongsin Poosankam, Kevin Zhijie Chen, Edward XueJun Wu, and Dawn Song. “MACE: Model-inference-Assisted Concolic Exploration for Protocol and Vulnerability Discovery.” In: *Proceedings of the 20th USENIX Security Symposium*. 2011.

[7] Seung-Seok Choi, Sung-Hyuk Cha, and Charles C. Tappert. “A Survey of Binary Similarity and Distance Measures”. In: *Journal of Systemics, Cybernetics and Informatics* 8.1 (2010).

[8] Weidong Cui, Jayanthkumar Kannan, and Helen J. Wang. “Discoverer: Automatic Protocol Reverse Engineering from Network Traces”. In: *Proceedings of 16th USENIX Security Symposium*. 2007.

[9] Julien Duchêne, Colas Le Guernic, Eric Alata, Vincent Nicomette, and Mohamed Kaâniche. “State of the Art of Network Protocol Reverse Engineering Tools”. In: *Journal of Computer Virology and Hacking Techniques* (Jan. 2017).

[10] Robert C. Edgar. “MUSCLE: Multiple Sequence Alignment with High Accuracy and High Throughput”. In: *Nucleic Acids Research* 32.5 (Mar. 2004).

[11] Hossein Fereidooni, Jiska Classen, Tom Spink, Paul Patras, Markus Miettinen, Ahmad-Reza Sadeghi, Matthias Hollick, and Mauro Conti. “Breaking Fitness Records without Moving: Reverse Engineering and Spoofing Fitbit”. In: *CoRR* abs/1706.09165 (June 2017). arXiv: 1706.09165.

[12] Hugo Gascon, Christian Wressnegger, Fabian Yamaguchi, Daniel Arp, and Konrad Rieck. “PULSAR: Stateful Black-Box Fuzzing of Proprietary Network Protocols”. In: *11th International Conference of Security and Privacy in Communication Networks, Revised Selected Papers*. SecureComm. Springer, 2015.

[13] Tammo Krueger, Hugo Gascon, Nicole Krämer, and Konrad Rieck. “Learning Stateful Models for Network Honeypots”. In: *Proceedings of the 5th ACM Workshop on Security and Artificial Intelligence*. ACM, 2012.

[14] Corrado Leita, Ken Mermoud, and Marc Dacier. “ScriptGen: An Automated Script Generation Tool for Honeyd”. In: *Proceedings of the 21st Annual Computer Security Applications Conference*. IEEE Computer Society, 2005.

[15] John Narayan, Sandeep K. Shukla, and T. Charles Clancy. “A Survey of Automatic Protocol Reverse Engineering Tools”. In: *ACM Computing Surveys* 48.3 (Dec. 2015).

[16] Jeremy Rauch. *Protocol Debug (PDB)*. Blackhat. Las Vegas, 2006.

[17] Robert R. Sokal and Charles D. Michener. “A Statistical Method for Evaluating Systematic Relationships”. In: *University of Kansas Scientific Bulletin* XXXVIII-2.22 (Mar. 1958), pp. 1409–1438.

[18] Lusheng Wang and Tao Jiang. “On the Complexity of Multiple Sequence Alignment”. In: *Journal of Computational Biology* 1.4 (1994).

[19] Yipeng Wang, Xiao-chun Yun, Muhammad Zubair Shafiq, Liyan Wang, Alex X. Liu, Zhibin Zhang, Danfeng Yao, Yongzheng Zhang, and Li Guo. “A Semantics Aware Approach to Automated Reverse Engineering Unknown Protocols”. In: *Proceedings of the 20th IEEE International Conference on Network Protocols*. IEEE Computer Society, 2012.

[20] Gilbert Wondracek, Paolo Milani Comparetti, Christopher Krügel, and Engin Kirda. “Automatic Network Protocol Analysis”. In: *Proceedings of the Network and Distributed System Security Symposium*. The Internet Society, 2008.

[21] Christian Wressnegger, Ansgar Kellner, and Konrad Rieck. “ZOE: Content-based Anomaly Detection for Industrial Control Systems”. In: *Proceedings of the 48th Conference on Dependable Systems and Networks*. 2018.

Appendix

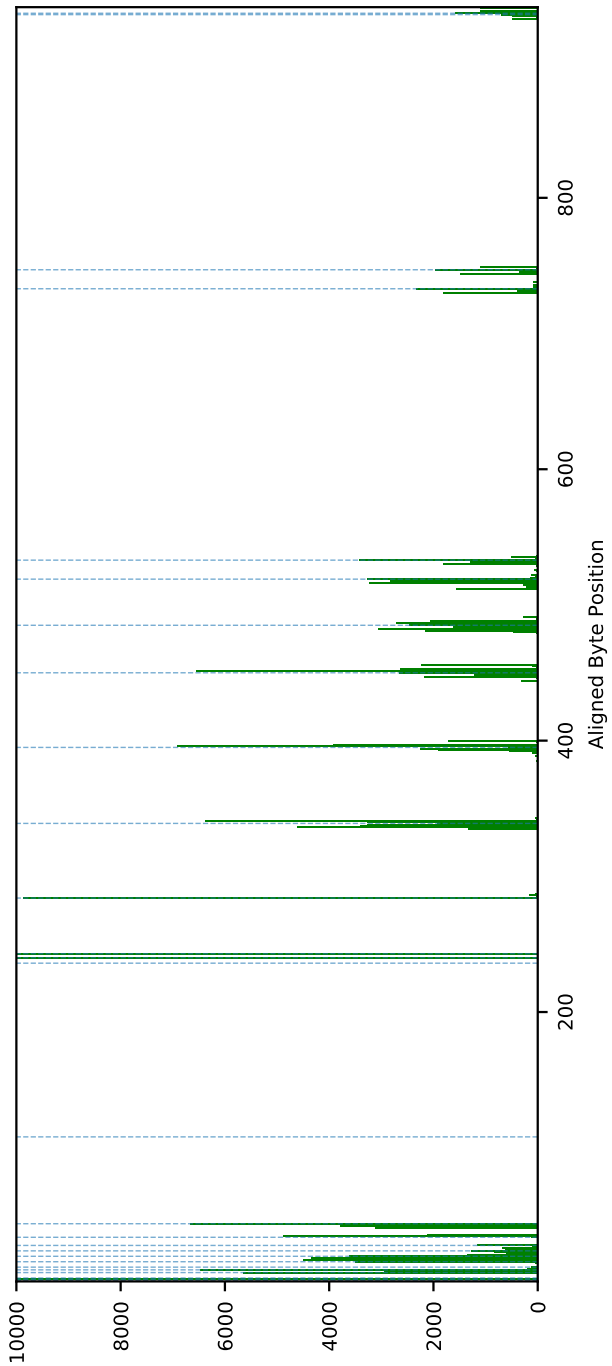


Figure 8: Histogram of inferred field boundaries around true field boundaries for 10 000 DHCP messages. The vertical dashed blue lines denote true boundaries; the bars count in how many messages a boundary was inferred at that byte position.

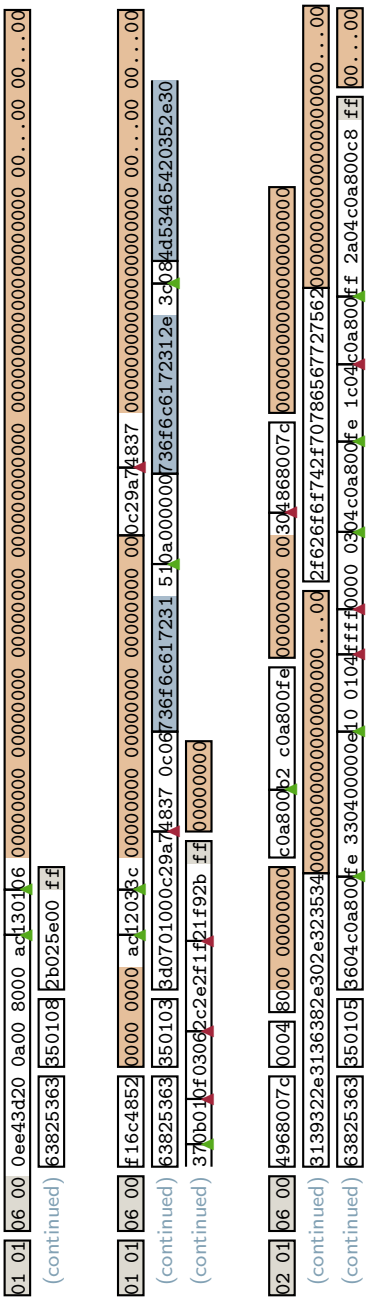


Figure 9: Example DHCP message segmentations annotated with the comparison to the dissector. True fields are separated by SPACEs; inferred fields are in framed boxes. Values are in hexadecimal notation. The color code of the marks is explained in Section 6.1.

Protocol	σ	Format Match Score		
		worst	average	best
DNS	0.9	0.021	0.362	0.695
	0.6	0.103	0.447	0.711
DHCP	0.9	0.115	0.433	0.701
	0.6	0.198	0.535	0.733
NTP	0.9	0.094	0.423	0.671
	1.2	0.074	0.453	0.672

Table 4: Comparison of the inference quality depending on parameter σ: Format inference results for worst, average, and best quality of 10 000 messages of each protocol inferred with default $\sigma = 0.9$ and the individual optimum for each protocol trace (see Section 6). The differences of these quality scores are visualized in Figure 10.

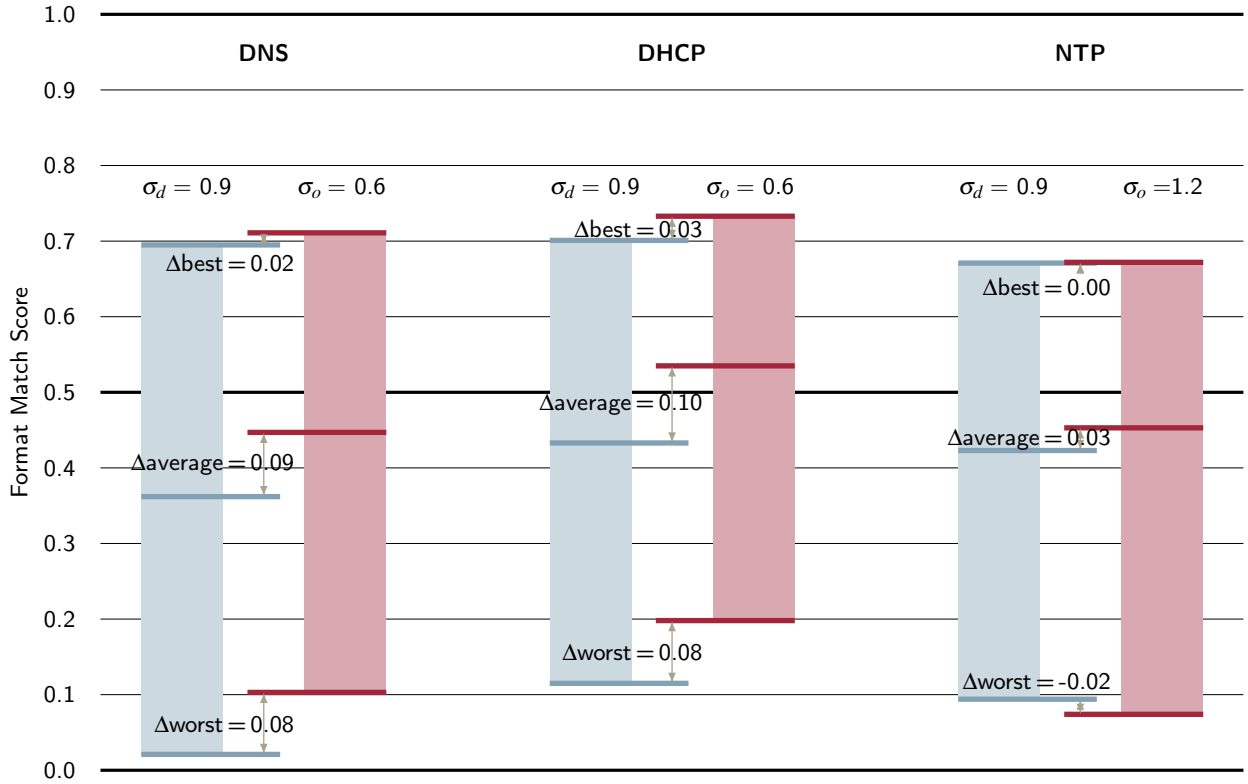


Figure 10: Comparison of the inference quality depending on parameter σ: the difference Δ between an inference with the proposed default of $\sigma_d = 0.9$ for unknown protocols and an inference with the individually optimal σ_o for each evaluated protocol trace as presented in Section 6; worst, average, and best refers to the FMS results as listed in Figure 10.