Message Type Identification of Binary Network Protocols using Continuous Segment Similarity

Stephan Kleber
Institute of Distributed Systems
Ulm University, Germany
stephan.kleber@uni-ulm.de

Rens W. van der Heijden
Institute of Distributed Systems
Ulm University, Germany
rensvdheijden@gmail.com

Frank Kargl
Institute of Distributed Systems
Ulm University, Germany
frank.kargl@uni-ulm.de

***Abstract*—Protocol reverse engineering based on traffic traces infers the behavior of unknown network protocols by analyzing observable network messages. To perform correct deduction of message semantics or behavior analysis, accurate message type identification is an essential first step. However, identifying message types is particularly difficult for binary protocols, whose structural features are hidden in their densely packed data representation. We leverage the intrinsic structural features of binary protocols and propose an accurate method for discriminating message types.**

Our approach uses a similarity measure with continuous value range by comparing feature vectors where vector elements correspond to the fields in a message, rather than discrete byte values. This enables a better recognition of structural patterns, which remain hidden when only exact value matches are considered. We combine Hirschberg alignment with DBSCAN as cluster algorithm to yield a novel inference mechanism. By applying novel autoconfiguration schemes, we do not require manually configured parameters for the analysis of an unknown protocol, as required by earlier approaches.

Results of our evaluations show that our approach has considerable advantages in message type identification result quality and also execution performance over previous approaches.

***Index Terms*—network reconnaissance; protocol reverse engineering; vulnerability research**

I. Introduction

Several recent surveys by Narayan *et al.* [1], Duchêne *et al.* [2], and Kleber *et al.* [3] describe the current state of the art for protocol reverse engineering based on network traffic traces or programs. In this paper, we focus on traffic analysis based on information gained by observing only the communication link. Such traffic analysis is non-invasive and does not require access to programs or control over any entity and therefore is regularly applied [4]. This method of network analysis has been used to gain comprehension of hitherto unknown network protocol, e. g., NetBios services [5], the Koobface command-and-control protocol [6], OSCAR [7], and IoT protocols [8]. The knowledge gained was then used for network analysis and security-relevant tasks, like vulnerability testing by fuzzing [7], the setup of honeypots [5], [6], analyzing botnets [6], and automated network modeling [8].

Protocol reverse engineering based on network traffic traces can roughly be divided into four steps, as shown in Fig. 1. First, the protocol traffic is recorded into traces during preprocessing. The second step is a structural analysis, whose purpose is to derive formats (e. g., data fields) of individual messages exchanged within the protocol. After analyzing the internal structure of messages, one important step is grouping these messages into message types (for DHCP, e. g., Discover, Offer, ACK). This information can then be used to derive a state machine that describes the grammar of the protocol. We place the derivation of the state machine out of the scope of our work, since adequate methods already exist for this analysis step [6], [9]–[11] that can be combined with our work. In this paper, we propose a novel approach to automate the task of message type identification for binary protocols.

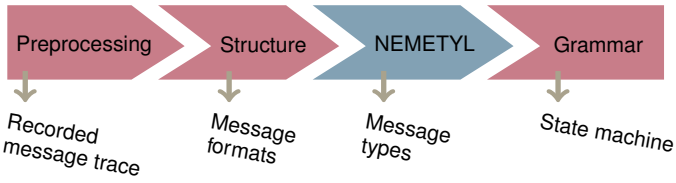


Fig. 1: Steps of network traffic trace analysis.

As the mentioned surveys show, current methods for traffic analysis mainly focus on textual protocols, which use separators and keywords that are discernible by natural language processing (NLP). Most binary network protocols, which represent data in concise bit patterns, lack these structural features required for NLP. Most of the few known methods tailored for binary protocols are derived from bio-informatics algorithms, e. g., Needleman-Wunsch [12]. When analyzing network protocols by these algorithms, messages commonly are interpreted as sequences of bytes. To work around the exponential complexity of naïve multiple sequence alignment [13], known applications for message analysis use the agglomerative clustering of UPGMA [14]. The algorithm recursively *agglomerates*, i. e. merges, similar pairs of clusters. This process builds a phylogenetic guide-tree with a complexity of the square of the amount of sequences [13].

In the bio-informatics use case, a phylogenetic tree is favorable, since it reflects the evolution of genome sequences. For protocol messages, no evolutionary relations exist, thus the computational overhead of agglomerative clustering is unnecessary. Moreover, aligning individual byte values detaches

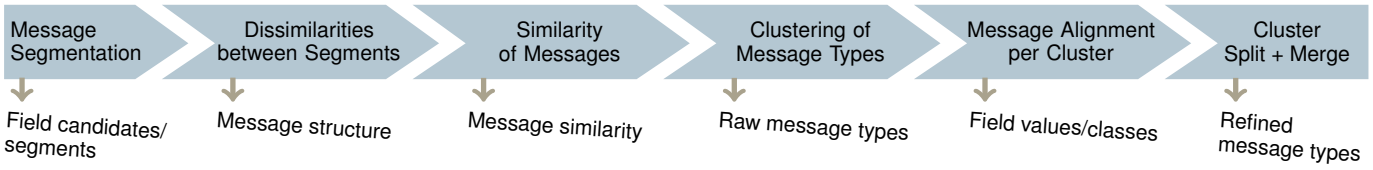


Fig. 2: Sequence of steps of NEMETYL and its intermediate results.

them from their context. This loss of context is undesirable since the exact same value cannot be unconditionally expected in multiple independent messages of binary protocols; this may lead to spurious relationships of values across messages.

In contrast, our segment-based approach is designed to provide better efficiency while improving the result quality at the same time. To accomplish this, instead of aligning byte-by-byte, we first split messages into segments using computationally cheap heuristics, like cutting them into fixed length chunks or applying a more advanced approach called NEMESYS [15]. Based on domain knowledge about the characteristics of typical network messages, we then derive message types from these segmented messages. Since we deal with unknown protocols, structure and data values are obscured. Thus, like previous approaches, we must rely on general domain knowledge about network protocols. We obtained this knowledge from analyzing numerous binary protocols in order to ensure our approach's general applicability. From this knowledge, we derive assumptions that allow us to design heuristic methods on which our approach relies. These heuristics do not rely on phylogenetic relations, enabling us to use a more efficient clustering algorithm.

The main contribution of our paper is a novel *message type identification* approach for unknown binary protocols. Unlike previous methods, our approach is designed to provide higher accuracy without manual parameter selection. To achieve this goal, we developed solutions to task-specific but fundamental challenges in the handling of sequential binary data. These contributions are the interpretation of *binary data as feature vectors*, a novel way to apply a *vector distance to unequally-sized feature vectors*, the application of *Hirschberg alignment* and the *DBSCAN cluster algorithm* to the area of protocol reverse engineering, and a novel method for *auto-configuring the DBSCAN parameter* ϵ. We implement and evaluate our approach in **NEMETYL**: **NE**twork **ME**ssage **TY**pe identification by a**L**ignment on similarity between message segment feature vectors. To evaluate NEMETYL we use known real-world binary protocols as a baseline. We determine the inference quality in terms of well-known cluster properties and compare them to the byte-wise sequence-alignment message inference performed by the state-of-the-art tool Netzob[1] [10].

In the next section, we discuss related work. In Sec. III, we present the details of our approach followed by a description of design decisions for its implementation NEMETYL in Sec. IV. Sec. V contains the results of our evaluation of NEMETYL. Finally, we outline our ideas for future work and conclude the paper in Sections VI and VII.

[1] github.com/netzob/netzob. All URLs last accessed on 18 Dec 2019.

II. Related Work

Static traffic analysis is a specific kind of protocol reverse engineering, where network traffic between two genuine entities is monitored purely passively, e.g., without injecting own messages or filtering some. The use of sequence alignment to perform such an analysis was first suggested by Beddoe [16]. Since Beddoe's paper, a variety of algorithms from natural language processing and bio-informatics have been applied to network protocols [3]. There are also practical implementations to perform static traffic analysis, the most versatile of which is Netzob [10].

ScriptGen [5], Discoverer [17], and Netzob all use sequence alignment to infer message types. ScriptGen and Discoverer differ from Netzob in that they align subsequences of messages (tokens or segments) instead of single bytes. They propose effective methods to identify such tokens in textual protocols. In addition, ScriptGen proposes the derivation of tokens from frequency, variance, and other byte characteristics throughout all messages of a trace. Although ScriptGen and Discoverer claim their methods to be universally applicable, they leave it to future work to solve the details of analyzing binary protocols. All presented methods are using agglomerative clustering by UPGMA [14], which prohibits the analysis of large traces and ignores byte-contexts as discussed in the introduction.

FieldHunter [18] combines concepts from Netzob, Discoverer, ScriptGen, and other related methods. It provides solutions for a number of challenges for format inference, like characterization of field types. However, FieldHunter is still based on byte-value features and thus also misses details related to message structure.

In related work, the coverage of the analysis of binary protocols is sparse and the robustness of existing solutions is limited due to the kind of byte-wise analyses they perform. Furthermore, the common combination of agglomerative clustering with conventional sequence alignment prohibits the analysis of large traces due to the high computational demand.

III. Approach

Our approach, NEMETYL, consists of the steps illustrated in Fig. 2. These steps are described in the following subsections. Among others, we require a suitable definition of similarity between segments (message subsequences) as well as between messages (message structure) and then use clustering

to identify messages with a similar structure. Finally, we refine these clustered message types to take into account differences between types that have the same structure.

A. Message Segmentation

As a prerequisite, NEMETYL relies on a segmentation of each message into atomic chunks of consecutive bytes. Ideally, such segments correspond to protocol fields, but for NEMETYL an approximate match of segments with field boundaries suffices. Thus, any heuristic method to obtain segments from messages may be applicable. In our current work, we investigate three different segmenters:

tshark generates segments from true fields determined by tshark's dissectors. This qualifies as ground truth for field boundaries of messages from already known protocols.

4-bytes-fixed uses fixed chunks of 4 byte length as segments. With no additional information available about a protocol to analyze, this is an extremely simple and efficient fallback that is always available.

NEMESYS is a message segmenter we proposed previously [15]. This heuristic and computationally cheap method approximates field boundaries for any unknown network protocol by information-theoretical metrics.

B. Dissimilarity between Segments

To quantify dissimilarity between segments, we first need a representation that describes the contents of each segment.

1) Extracting Feature Vectors from Message Segment Bytes: We generate feature vectors from each segment's byte values, use hexadecimal notation for bytes, e. g., $b = \texttt{0xA4}$, and write a segment s as an ordered set of n byte values $b_0, \ldots, b_{n-1}$: $s = \langle b_i \rangle$, with $i \in [0, n-1]$

We interpret the sequence of byte values $\langle b_i \rangle$ of a segment as a feature vector $\mathbf{s}$, where the ith vector component $\mathbf{s}_i = b_i$. For example, given a segment $s = \langle b_i \rangle = \langle \texttt{0x17, 0x23, 0x00, 0x42} \rangle$, the feature vector is written as $\mathbf{s} = (\texttt{0x17}, \texttt{0x23}, \texttt{0x00}, \texttt{0x42})^T$.

2) Canberra Dissimilarity: We use the Canberra distance [19] to quantify the dissimilarity between segments, which for vectors u and v of equal dimension n, is defined as:

$$d_C(u, v) = \sum_{i=0}^{n-1} \frac{|u_i - v_i|}{|u_i| + |v_i|}, \tag{1}$$

The Canberra distance correlates well with the intuitive similarity of byte sequences in that it relates different values in sequences to their common mean of values. However, our application as a measure of dissimilarity requires to extend this distance to be applicable to vectors of differing dimensionality; otherwise segments of unequal length would not be comparable. To accomplish this, we *generalize the notion of the Canberra distance to vectors of different length* by choosing an embedding of the higher-dimensional vector into the lower-dimensional space. We normalize the distances from the embedded vectors to assure comparability across vector spaces. Note that this generalization is only valid for our specific application in which the vector represents a segment, as it violates the triangle inequality, and therefore constitutes not a distance anymore, but instead a dissimilarity. We call this the *Canberra dissimilarity* d_m.

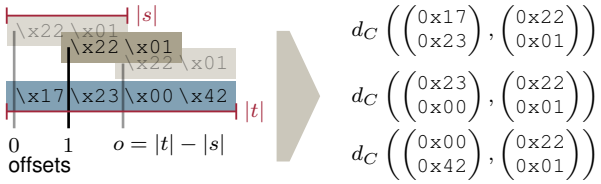


Fig. 3: Sliding window interpretation of linear subspaces.

Without loss of generality, let s and t be segments with $|s| \leq |t|$ and generate the associated vectors $s \to \mathbf{s}$, $t \to \mathbf{t}$. The lengths of the segments imply the dimensionality of the associated vectors: $|s| = \dim \mathbf{s}$ and $|t| = \dim \mathbf{t}$. For the case where $|s| = |t|$, we define $d_m(\mathbf{s}, \mathbf{t}) = \frac{d_C(\mathbf{s},\mathbf{t})}{|s|}$. For the case where $|s| < |t|$, we apply the following procedure to derive the Canberra dissimilarity.

First, using a sliding window approach, we derive a series of possible embeddings of the longer feature vector $\mathbf{t}$ into the vector space of the shorter feature vector. The intuition behind this is illustrated in Fig. 3. This shortened vector $\mathbf{t}'$ can then be compared using the standard Canberra distance, $d_C(\mathbf{t}', \mathbf{s})$. The sliding window of length $|s| = \dim \mathbf{s}$ can be represented using an offset o as $\mathbf{t}_{[o,o+|s|]}$. Using this notation, the set of suitable embeddings is defined as:

$$T = \cup_{o=0}^{|t|-|s|} \{\mathbf{t}_{[o,o+|s|]}\} \tag{2}$$

Second, we define the minimum Canberra dissimilarity d_β between a set of vectors T as defined above and a shorter candidate vector $\mathbf{s}$ as follows:

$$d_\beta(T, \mathbf{s}) = \frac{\min_T(\{d_C(\mathbf{t}_{[o,o+|s|]})\})}{|s|} \tag{3}$$

This selects the lowest Canberra distance as a dissimilarity between the longer and the shorter segment. Essentially, we derive the nearest candidate vector (in terms of d_C). We then normalize d_β to $|s|$ in accordance with the explanation above.

Third, we note that for use as a dissimilarity, the pure d_β provides undesirable results, since the difference in segment length is completely ignored. Without the loss of generality, we therefore extend our previously defined dissimilarity d_m with a non-linear modification based on the relative length of the segments. Thus, we redefine[2] d_m as follows:

$$d_m(\mathbf{s}, \mathbf{t}) = \underbrace{\frac{|s|}{|t|} d_\beta(\mathbf{s}, \mathbf{t})}_{\text{subterm 1}} + \underbrace{r}_{\text{subterm 2}} + \underbrace{(1 - d_\beta(\mathbf{s}, \mathbf{t}))r(\frac{|s|}{|t|^2} - p_f)}_{\text{subterm 3}} \tag{4}$$

with $r = \frac{|t|-|s|}{|t|}$ representing the length difference between the segments, and with a parameter p_f to set the non-linear penalty in subterm 3, as discussed below.

[2]Our previous definition of d_m still is valid as defined before, since $d_\beta(\mathbf{s}, \mathbf{t}) = \frac{d_C(\mathbf{s},\mathbf{t})}{|s|}$ and $r = 0$ for $|s| = |t|$.

The first subterm normalizes the minimum Canberra dissimilarity d_β such that the longer segment's length $|t|$ becomes the reference. This represents the immediate dissimilarity between the selected components of the longer vector and all components of the shorter one. As the dimensionality difference increases, more information is discarded from $\mathbf{t}$ in the computation of d_β. In Table I, example 1 illustrates that d_β erroneously assigns a distance of zero if the shorter vector is contained in the larger vector. To correct for this, we use the second subterm, a linear factor r. Example 2 in Table I illustrates that d_m attests two vectors with a small d_β to have a dissimilarity just below their relative dimensionality difference ($\frac{1}{2}$). Example 3 shows how this influences longer segment pairs that have the same relative dimensionality differences. Finally, if a longer absolute part of t is used to calculate d_m, this should result in a lower dissimilarity of the segments, since we need to make less assumptions about the missing vector components. For example, if 4 out of 5 components are selected by d_β, the final dissimilarity d_m should be higher than if 16 of 20 components are known, even though the fraction is the same. Intuitively, it is more unlikely that 16 components mismatch for any two similar segments than 4. This absolute difference between the vectors' dimensionalities is taken into account by the third subterm, which non-linearly increases to penalize larger absolute dimensionality differences. The factor p_f parameterizes the steepness of the slope in the penalty increase relative to d_β.

Each of the subterms describes a disjoined aspect of dissimilarity between two segments and we can, thus, connect them by simple summation. The subterms representing these aspects sum up to a value of 1 for two maximally different segments. d_m decreases for more similar segments and reaches 0 for identical ones.

3) Refining Dissimilarities of Char Sequences: Segments consisting of sequences of characters are special in a way that make the raw Canberra dissimilarity values between vectors for char sequences unsuitable. We therefore reduce the dissimilarity values of pairs of segments that both are chars by a factor of 0.5 to reflect the characteristics of char sequences. We explain our heuristic to identify char sequences in the implementation description (Sec. IV-B).

C. Similarity of Messages

In this step, we analyze the structural similarity of messages using the Needleman-Wunsch (NW) algorithm, which is widely used for this purpose in protocol reverse engineering. The algorithm finds the optimal alignment of two strings using a substitution matrix $\mathbf{N}$ of possible alignments and the associated alignment score. The maximum of these scores is the most suitable alignment. The score is computed based on a gap penalty p_g and a similarity matrix $\mathbf{S}$ that expresses the similarity of segments. The gap penalty is used to penalize gaps (i. e., empty segments added to one of the messages) in the alignment. The similarity between segments expresses whether they accurately align. Most authors [5], [10], [16], [17] simplify the algorithm by choosing a similarity matrix such that the diagonal contains a match value p_m and all other values are p_d. This makes alignment a boolean decision, which is a major drawback of existing alignment methods [16], as similar but unequal positions cannot be quantified appropriately. In this section, we describe how the segment similarity of the previous section can be used to improve the alignment of NW. We then use the NW score to define message similarity, from which we derive a message **dis**similarity d_{NW} that we use in the next step of NEMETYL.

TABLE I: Examples demonstrating the differences between minimum Canberra dissimilarity and Canberra dissimilarity.

Example	1	2	3	4
$\mathbf{t}$	0x0008	0x0208	0x5706906e	0x0208
$\mathbf{t}_{[o,\lvert s\rvert]}$	0x00	0x08	0x5706	0x0208
$\mathbf{s}$	0x00	0x07	0x2700	0x0008
d_β	0.000	0.067	0.690	d_m ($\lvert s\rvert = \lvert t\rvert$)
d_m	0.460	0.496	0.814	0.5

As an example, we use the two messages m_0 = 0x0208000807 and m_1 = 0x07270000082317. First, we need to derive the similarity matrix S required for NW. This is done by computing the pairwise similarity as $1-d_m$, where d_m is the dissimilarity measure from the previous step. Without loss of generality, we define that p_d is the minimum similarity (zero) and p_m is the maximum similarity (one). Intuitively, the pairwise similarity represents a spectrum of similarity, rather than the binary decision made by previous work. For this example, this results in the following symmetric matrix:

S	07	2700	2317	0208	0008
07	1.00	0.16	0.25	0.50	0.50
2700	0.16	1.00	0.47	0.05	0.00
2317	0.25	0.47	1.00	0.31	0.26
0208	0.50	0.05	0.31	1.00	0.50
0008	0.50	0.00	0.26	0.50	1.00

We use this matrix together with a gap penalty $p_g = -1$ to align the messages using NW. This algorithm computes a substitution matrix $\mathbf{N}$ based on $\mathbf{S}$ and p_g, which for this example is:

N		0208	0008	07
	0	−1	−2	−3
07	−1	0.50	−0.50	−1.00
2700	−2	−0.50	0.50	−0.33
0008	−3	−1.50	−0.50	1.01
2317	−4	−2.50	−0.50	**0.76**

The NW score $\mathbf{N}_{[i,j]}$ in the substitution matrix is the degree of congruence of a specific pair of segments, namely the $i-1$th segment of m_0 (columns) and the $j-1$th of m_1 (rows). Since the NW algorithm starts with blank strings, the dimensions of the matrix $\mathbf{N}$ are $|m_0|_s+1$ columns and $|m_1|_s+1$ rows, where $|\cdot|_s$ refers to the number of *segments* with the matrix' indexing starting from 0. This matrix allows us to define the message similarity based on the NW score. We follow Smith *et al.* [20], and define the NW score $\mathfrak{N}(m_0, m_1)$ as the bottom-right entry in the substitution matrix $\mathbf{N}$, i.e., $\mathfrak{N}(m_0, m_1) = \mathbf{N}_{|\mathbf{m_0}|_s,|\mathbf{m_1}|_s}$. The self-similarity of each message is the amount of segments, since in that case, each segment has a similarity of 1. The message similarity can thus be described as follows:

	m_0	m_1
m_0	4	0.76
m_1	0.76	3

NW scores quantify the similarity of messages through a value in $\mathbb{R}$. However, our goal is to find groups of messages that are similar, for which an everywhere-positive dissimilarity function is required. We thus define a function $d_{\text{NW}} : M \times M \to \mathbb{R}^+$ based on the message similarity derived from the NW score $\mathfrak{N}(m_0, m_1)$:

$$d_{\text{NW}}(m_0, m_1) = 1 - \frac{\mathfrak{N}(m_0, m_1) - \mathfrak{t}_{\min}(m_0, m_1)}{\mathfrak{t}_{\max}(m_0, m_1) - \mathfrak{t}_{\min}(m_0, m_1)} \quad (5)$$

with $\mathfrak{t}_{\min}(m_0, m_1) = \min(|m_0|_s, |m_1|_s) \cdot \min(p_g, p_m, p_d)$ and $\mathfrak{t}_{\max}(m_0, m_1) = \min(|m_0|_s, |m_1|_s) \cdot \max(p_g, p_m, p_d)$. Recall p_m and p_d are bounds on segment similarity. Therefore, this equation essentially normalizes the NW scores to a similarity value, which is then subtracted from 1 to gain a dissimilarity.

By the end of this step, we have generated a message dissimilarity matrix that contains an entry for the dissimilarity between each pair of messages.

D. Clustering of Message Types

The message dissimilarity matrix from the previous step can directly be used as input to cluster messages into types. The challenge is to automatically configure the parameters of the cluster method in a suitable way. Our approach aims for a method that requires no a-priori knowledge of the protocol. We identified DBSCAN [21] to be suitable for this task. It is a density-based method and therefore makes no assumptions about the shape of clusters. DBSCAN requires no target number of clusters as parameter and deals with noise by rather not assigning a sample to any cluster than choosing the wrong one. Applying clustering without prior knowledge about the trace requires to automatically derive the necessary parameters for DBSCAN from the trace itself.

DBSCAN has two parameters, min_samples and ϵ, which are auto-configurable since they can be derived from statistical properties of the trace with a high rate of success. The first, min_samples, determines the smallest valid cluster (see Sec. IV-D). The second parameter, ϵ, defines a range around a density core of samples that should constitute a cluster. This parameter is specific to the data being clustered; in our case, it depends on the characteristics of a trace. If ϵ is too large, random noise will be placed in clusters, while if ϵ is too small, meaningful clusters with only few samples will be considered noise. As suitable values differ per trace, we developed an auto-configuration technique for this parameter. Our choice for ϵ is based on the distance to the kth nearest neighbor, measured by NW-score dissimilarity d_{NW} between messages. The intuition is that as this function typically shows a sudden change for a message set with well-defined clusters, the choice for ϵ should be exactly at this point. The best choice for ϵ for the set of messages M in a trace thus depends on k, which is configured in the following.

We first define a family of functions $f_k : M \to \mathbb{R}$ of which the kth function maps each message to the NW-score dissimilarity of the kth nearest neighbor of that message. We refer to this as the kth nearest neighbor distance function, which quantifies how sparse a potential cluster of size k is. More formally,

$$f_k(m) = \max(D \subset \{d_{\text{NW}}(m, m_k) : m_k \in M\}, \quad |D| = k, \text{s.t.} \sum_{a \in D} a \text{ is minimal}) \quad (6)$$

We then apply a Gaussian filter (with parameter σ, see IV-D) and determine the inputs (k, m) such that the curvature of the k nearest neighbor distance function is maximal. This process is biased towards large k, which we compensate for by dividing through the value of the function, more formally:

$$\arg\max_{(k,m)} \frac{G(f_k''(m))}{G(f_k(m))} \quad (7)$$

As f_k is a family of discrete functions, *curvature* is here defined using the forward difference, i. e., $f_k''(x) = f_k'(x+1) - f_k'(x)$ and $f_k'(x) = f_k(x+1) - f_k(x)$. In other words, we find the m_ϵ and function f_{k_ϵ} for which the largest change in curvature is observed. We then set ϵ to the value of the chosen function at the chosen message, i. e., $\epsilon = f_{k_\epsilon}(m_\epsilon)$.

Finally, we cluster messages into types by DBSCAN using the auto-configured epsilon and the message dissimilarity matrix of all $d_{\text{NW}}(m_i, m_j)$ as input. This step yields highly precise clusters of messages classified into message types.

E. Message Alignment per Cluster

To determine the common internal structure for each message type, we align the messages within each cluster to the cluster's medoid, i. e., the message with the lowest dissimilarities to all other messages within the cluster.

This step results in clusters that each are commonly aligned on the segments of their messages. Each common position in the alignment of all segments we call a "field candidate".

F. Cluster Refinement

The steps up to this point – segmentation, calculation of segment dissimilarity and message similarity, clustering, and alignment – establish the structural similarities of messages and classify them accordingly. Different message types that are structurally identical but distinguished only by dedicated field values, e. g., message type A is denoted by value `0x01` and type B by value `0x02` in field 1, cannot be discriminated by this process alone. Therefore, we refine the clustering output by applying additional heuristics to find such cases, splitting underspecific clusters and merging overspecific clusters, to find the message types more accurately. We discuss the details of these heuristics in Sec. IV-F.

1) Splitting Underspecific Clusters: To identify dedicated field values that distinguish structurally identical message types, we search for fields in the aligned clusters that contain only few distinct values. These fields are likely to denote individual message types within one cluster and we therefore split it into sub-clusters that each contain only messages having one of the distinct values in this specific field.

2) Merging Overspecific Clusters: For some structurally complex protocol messages, multiple clusters exist for a single message type. We merge these overspecific clusters of similarly structured messages into one. Thus, the alignment of each cluster is generalized into dynamic, static value (e. g., `0100`), and `GAP` field candidates. Two clusters that exhibit a compatible structure according to the generalized alignment of the clusters are merged. An example of the aligned field candidates for a mergeable pair of clusters looks like:

```
DYNAMIC   0100   0001   DYNAMIC   DYNAMIC   0001
DYNAMIC   0100   GAP      00       001c     0001
```

The final result of our approach are clusters of messages with similar structure that have an increased accuracy compared to the result of the raw clustering.

IV. NEMETYL Implementation

We implemented a modular proof of concept of NEMETYL to validate our approach. It uses numpy[3], scipy[4], sklearn.cluster.DBSCAN[5], netzob[6], and NEMESYS[7]. This section discusses parameter choices and design decisions made as part of the implementation and also highlights some more details on the general process.

A. Message Segmentation

We investigate three different segmentation approaches: *tshark*, *4-bytes-fixed*, and *NEMESYS*. The *tshark* approach uses the true field boundaries and is thus not available for unknown protocols but serves as a baseline for perfect segment knowledge. *4-bytes-fixed* ignores real field boundaries and uses 4-byte chunks as segments.

Finally, *NEMESYS* approximates the true field boundaries without knowledge of the true structure. NEMESYS compares subsequent bytes with a method called "Gaussian-filtered Bit Congruence deltas". Throughout an individual message, this method reveals value patterns that it exploits to derive probable field boundaries. In order to apply NEMESYS in NEMETYL, we needed to make three small modifications to the original scheme. First, we additionally perform a frequency analysis of the most common segment values. If one of these most frequent values is a subsequence of another, larger segment, we split this larger segment to reveal the more frequent and therefore more probable segment boundary. Second, we enhance NEMESYS by our char sequence detection to improve the recognition of sequences of textual parts within the binary protocol. Third, NEMESYS has issues with the initial segment of each message. Therefore, we split each message's first segment into dedicated, one-byte segments. Due to its heuristic nature, NEMESYS leads to less distinct message-type cluster borders than the tshark segmenter. To compensate, we reduce the allowed range around density-cores for the DBSCAN-based clustering relative to the automatically determined value: Therefore when using NEMESYS, we adjust ϵ by an additional factor of 0.8.

[3]www.numpy.org, version 1.13.3
[4]www.scipy.org, version 1.0.0
[5]scikit-learn.org/0.19/modules/generated/sklearn.cluster.DBSCAN.html
[6]github.com/netzob/netzob/tree/next, "next" branch
[7]github.com/vs-uulm/nemesys

B. Dissimilarities between Segments

As explained, we need to treat char sequences in a special way. However, since our approach is intended for the analysis of unknown protocols, we have no definitive knowledge about where to expect char sequences. We empirically determined characteristics of char sequences in 40 000 messages of the protocols DNS, NBNS, SMB, and DHCP from real-world traces. Thereby, we selected optimal values for a number of parameters that describe these characteristics to achieve a low number of false positives in the char detection. Based on this analysis, we detect hypothetical char segments if all of the following conditions are true:

- All bytes have values lower than `0x7f`; *(and)*
- the sequence is at least 6 bytes long; *(and)*
- the mean of byte values is between the thresholds t_l and t_h. We exclude zero bytes from the mean calculation to account for termination, padding, and latin chars in UTF-16. Our analysis determined working values to be $(t_l, t_h) = (50, 115)$; *(and)*
- the ratio of non-printable and non-zero chars in one segment to the segment's length is below 0.33.

We further validated the char detection by applying this heuristic to NTP, which contains no chars, showing no false positives.

C. Similarity of Messages

To compute NW scores, we use the Hirschberg alignment of the segments in a message pair. Hirschberg alignment [22] is a memory optimized version of Needleman-Wunsch [12]. Our approach requires that a gap is only inserted in the alignment if no similar segments are available at one position. Gaps therefore are scored negative and the value for the alignment parameter $p_g = -1$ was iteratively determined in a pilot study.

D. Clustering of Message Types

To robustly handle noisy data for the auto-configuration of DBSCAN's ϵ parameter, we smooth the values of the discrete message distance function by a Gaussian filter $G(d(s_0, s_k))$. The filter's single parameter is the standard deviation σ. A pilot study showed that $\sigma = \ln(n)$ with n being the number of neighbors removes noise sufficiently while retaining enough detail in the distance function. We further limit the impact of

noise in the dissimilarity values when selecting the distance function $f_k(m)$ by two actions: First, we limit the iteration of k to the closest 10 % of k-nearest neighbors. Secondly, to remove boundary effects for values of m close to the edges of the discrete distance function $f_k(m)$, we limit the range of m to $2\sigma < m < n - 2\sigma$ after applying the Gaussian filter.

Since we merge overspecific clusters in the final step, we set DBSCAN's min_samples parameter to the low value of 3.

E. Message Alignment per Cluster

For the actual alignment of messages within a cluster, we reuse the fixed alignment parameters from Sec. IV-C. First, we ascendingly sort the messages in a cluster by their dissimilarity to the medoid. Then we iterate through this list and align the next message to the common alignment of the medoid and all messages aligned in the previous iteration. This constitutes a progressive alignment of the previous messages with the next one. In each iterative alignment we introduce new gaps in all already aligned messages including the medoid.

F. Cluster Refinement

1) Splitting Underspecific Clusters: To discriminate multiple message types within one cluster of structurally similar messages, we first need to determine which fields distinguish the message type. As criterion, we use the observation that distinct values are frequent within one field that distinguishes the message type. As a heuristic, $t = \lfloor \ln(|c|) \rfloor$ is a threshold for the cluster c to define a frequent value among $|c|$ values. A field then is distinguishing message types if it contains only frequent values, determined by satisfying $\min(a) > t \geq |a|$ with a being the sorted set of value counts of the field. For example, if the values of one field in cluster c are 5 times `85`, `01` is contained 8 times, and `23` 9 times, then $a = \{5, 8, 9\}$, $\min(a) = 5$, $|a| = 3$, and $|c| = \sum a = 22$. Therefore, t is 3, and the inequality holds so that we consider this field to contain only frequent values. Thus, we found a field that likely distinguishes message types. All clusters will be split so that fields with frequent values will be the only value in the distinguishing field in a single cluster.

2) Merging Overspecific Clusters: To determine if alignments are similar enough to merge clusters, we abstract the alignments: One sequence of abstracted field candidates represents the collective structure of the messages in a cluster. We align these abstracted message structures of each pair of clusters by NW. Two clusters are merged if at least one of the following is true for all aligned field candidates:

- A gap is present in either structure at one position.
- Both field candidates have the same static byte values or both are dynamic.
- Either field candidate consists of only zero bytes.
- One field candidate is static, the other dynamic and contains the static candidate's value in at least one message.

V. Evaluation

We evaluate the quality of our message type identification approach by examining binary protocol traces of DNS, NBNS, NTP, SMB, and DHCP[8] and analyzing the precision P and recall R of identified message clusters, which are defined as:

$$P = \frac{\text{TP}}{\text{TP} + \text{FP}} \quad \text{and} \quad R = \frac{\text{TP}}{\text{TP} + \text{FN}}$$

For multiple clusters, TP, FP, TN, and FN, constituting the so called confusion matrix, are defined through the correct and incorrect assignments of messages, as discussed by Manning *et al.* [23]. Positives (_P) therefore are all pairs of messages that are classified into the same cluster, while such a pair is true (TP) if both messages are of the same type. Negatives (_N) and false (F_) assignments are defined accordingly. The amount of positives and negatives for n clusters c_i are thus given as:

$$\text{TP+FP} = \sum_i \binom{|c_i|}{2} \qquad \text{and} \qquad \text{TN+FN} = \sum_{i,j} \left(|c_i| \cdot |c_j|\right),$$

where $j = \{0 \dots (n-1)\} \setminus i$. The number of true positives and false negatives (and, through the above equations, the complete confusion matrix) are given by:

$$\text{TP} = \sum_i \sum_l \binom{|t_{i,l}|}{2} \text{ and } \text{FN} = \sum_i \sum_l \frac{(|t_l| - |t_{i,l}|) \cdot |t_{i,l}|}{2},$$

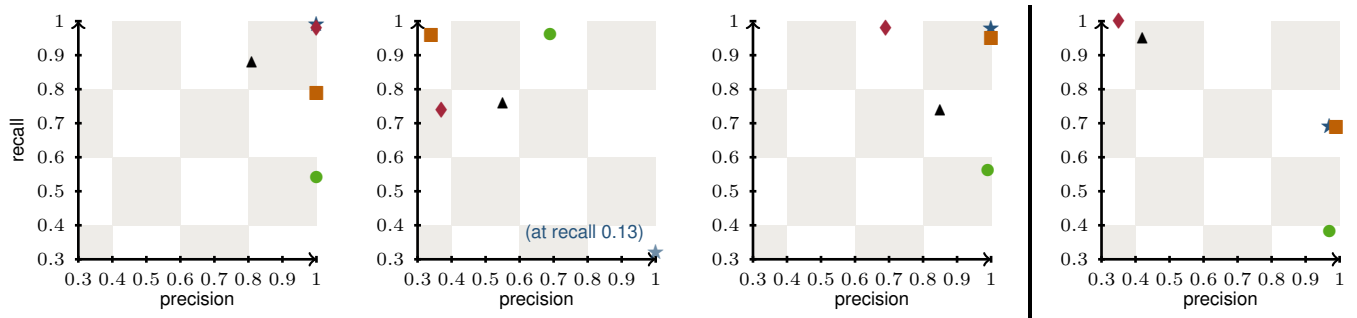
where $t_{i,l}$ are the messages of type l in cluster i, while t_l denotes the messages of type l and thus $t_l = \cup_i \, t_{i,l}$.

To obtain the required ground truth for testing our results against, we implemented Python modules that obtain the true field dissection of each message and compare these to the inferred message types. As ground truth about the protocol specification, we utilize tshark[9]. For each message in the trace, we compare the inference results to the according protocol dissector provided by tshark. As specimens, we use the binary protocols DNS, NBNS, NTP, SMB, and DHCP. We chose these protocols as representatives of different typical binary protocols. DHCP and SMB have 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 and NBNS contain mostly 2 byte binary fields mixed with variable length fields of ASCII-encoded characters. The traces we analyzed are publicly available[10]; we pre-processed each raw trace by filtering for only the desired protocol, removing duplicates of the payload, and truncating them to 1 000 messages each.

[8]Dynamic Host Configuration Protocol (RFC 2131), Domain Name System (RFC 1035), NetBIOS Name Service (RFC 1002), Network Time Protocol (RFC 958), and Server Message Block (Microsoft)

[9]Command line interface of Wireshark, see www.wireshark.org

[10]NTP, NBNS, SMB, and DHCP filtered from download.netresec.com/pcap/smia-2011/; DNS filtered from ictf.cs.ucsb.edu/archive/2010/dumps/ictf2010pcap.tar.gz



(a) Segmenting with tshark. (b) Fixed segments of 4 bytes. (c) Segments of NEMESYS. (d) Clustering quality of Netzob.

Fig. 4: Clustering quality results of NEMETYL using different segmenters and Netzob for comparison. The protocols are ● DHCP, ♦ DNS, ★ NBNS, ■ NTP, and ▲ SMB.

A. Test Cases for Clustering Quality

To show the validity of the assumptions that our approach is based on and the impact of the selected segmenter providing the field candidates, we run our implementation with different segmenters that provide all messages of a trace segmented into (1) the true fields from the dissector, (2) equal-length chunks of 4 bytes, and (3) the hypothetical fields inferred by NEMESYS.

B. Results

In this section, we present the results of applying NEMETYL with the three described segmenters. Our approach considers messages as noise if the clustering and refinement cannot determine a reliable choice for a classification. The amount of noise for each inference is given alongside with the precision and recall. To set our results into context, we use Netzob for inferring the same traces as we evaluate our approach with. Netzob uses NW to byte-wise align all messages and identify message types from hierarchical clustering using the scores for this alignment.

Netzob and the segmenter NEMESYS each require to set a parameter. The parameter σ adjusts the Gaussian filter in NEMESYS to the expected maximum lengths of fields in the target protocol. We used the optimal values given in Kleber *et al.* [15]. Netzob's parameter is the similarity threshold that needs to be reached for the hierarchical clustering to terminate and thus directly adjusts the clusters. For Netzob, we determined the optimal similarity thresholds for each protocol in multiple test runs iterating the parameter. The selected parameter values are given in Table II.

1) tshark Segments: Using true field borders as message segments, the message types of all protocols could be determined with high accuracy as can be seen in Table II, *tshark* column, and illustrated by Fig. 4a, with the exception of DHCP, which has a complex structure that does not clearly reflect the message type. This is noticeable by the low recall value, which is due to structurally diverse message types that are split up into multiple clusters. SMB's relatively low precision is due to one large cluster of structurally identical messages which our splitting heuristic was not able to discriminate further.

2) 4-bytes-fixed Segments: As to be expected, the use of an uninformed segmentation to find field candidates yields considerably worse results, as can be observed in Table II, *4-bytes-fixed* column, and is illustrated by Fig. 4b. For protocols that determine their message type by single flag fields, in this case NBNS, this causes low recall values. The simple field structure of DNS and NTP is obscured by the fixed segment splitting, leading to low precision values. However, especially for the complex protocols DHCP and SMB the similarities of the messages are sufficiently recovered by this simple method to identify significant differences in message types.

3) NEMESYS Segmens: Finally, NEMESYS infers field boundaries heuristically. In Table II, *NEMESYS* column, we give the quality and the sigma values used for each protocol to configure NEMESYS. We illustrate the results by Fig. 4c. It can be noticed that DHCP and SMB have comparably low recall results. This is due to the complexity of the protocol that not always reflects message types by their structure. Like for the tshark segmenter, more than one cluster is created for one message type, although the other types are identified correctly. The high correctness of the field boundaries determined by NEMESYS for NBNS and NTP causes the excellent quality of these protocols' inference. NEMESYS even outperforms tshark regarding NTP recall due to one large cluster that our heuristic splitting severs falsely in the case of tshark.

4) Netzob Baseline: For comparison, we apply the inference of Netzob to the same protocol traces as our approach. The resulting clusters' precision and recall, as illustrated in Fig. 4d, can be found in the last column of Table II, alongside the similarity threshold used for each protocol.

Unsurprisingly, NEMETYL in conjunction with the ground truth segmenter tshark clearly outperforms Netzob. In a more realistic case where the message segmentation is unknown, NEMETYL with the NEMESYS segmenter yields a lower recall for SMB compared to Netzob and significantly outperforms Netzob for DHCP, DNS, NBNS, NTP, and SMB in

TABLE II: Clustering quality of NEMETYL using different segmenters (columns two to four) and Netzob (column five).

Segmenter	*tshark*			*4-bytes-fixed*			*NEMESYS*				Netzob method		
	precision	recall	noise	precision	recall	noise	precision	recall	noise	σ	precision	recall	threshold
DHCP	1.00	0.54	35	0.69	0.96	0	0.99	0.56	7	0.6	0.97	0.38	78
DNS	1.00	0.98	18	0.37	0.74	17	0.69	0.98	7	0.6	0.35	1.00	50
NBNS	1.00	0.99	21	1.00	0.13	21	1.00	0.98	13	0.9	0.97	0.69	58
NTP	1.00	0.79	6	0.34	0.96	2	1.00	0.95	27	1.2	0.99	0.69	57
SMB	0.81	0.88	8	0.55	0.76	16	0.85	0.74	65	1.2	0.42	0.95	55

terms of precision. In summary, NEMETYL in combination with NEMESYS constitutes a relatively reliable method of high-quality to identify unknown protocol's message types that exhibit structural differences.

C. Interpretation

What sets NEMETYL apart from previous methods of message type identification is shown in the following interpretation of the evaluation results.

(1) The high quality of applying our implementation to dissector-generated true fields (Sec. V-B1) shows:

- Distinctive differences between field types exist and can be exploited to derive similarity between messages.
- The feature (byte value vector) and dissimilarity measure (based on Canberra) we employ works well in revealing these differences.
- Our interpretation of mixed-dimension vectors reflects real similarities between mixed-length segments.
- Our combination of alignment with a continuous, i. e., non-discrete, segment-similarity matrix for calculating alignment-match scores constitutes a valid method.

(2) Applying our implementation to field-structure-agnostic equal-length segments of messages (Sec. V-B2) shows:

- Our approach works to some extent even when no knowledge about field boundaries is available.
- Having fixed-length segments allows for a simple and efficient dissimilarity calculation providing quick results, especially for complex protocols.
- The dissector-generated fields for the tshark and NEMESYS segmenters required us to conceive a novel method to calculate dissimilarities between mixed-length segments. In contrast, using fixed lengths of segments in this second stage of the evaluation allows us to calculate distances by the well-known Canberra distance within the constant vector space of four dimensions. Comparing these alignment results to those of the tshark and NEMESYS segmenters, which use our Canberra dissimilarity, shows that Canberra dissimilarity provides valid dissimilarity measurements and thereby backs the validity of the mixed-length dissimilarity.

(3) Applying our implementation to NEMESYS-inferred segments of messages that denote hypothetical fields (Sec. V-B3) shows:

- The quality of the segmentation influences the alignment and type identification.
- Non-trivial segmentation is possible without a-priori knowledge about the protocol, in turn reinforcing the results of our previous work NEMESYS.
- Thus, for a fully automated tool, NEMESYS and NEMETYL are a feasible combination of methods.

D. 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 [2]. For example, a Man-in-the-Middle between two genuine entities can record the decrypted messages in transit [24]. This method requires control over the network topology.

Depending on the protocol trace to be analyzed, in particular complex message structures lead to overspecific clusters. Our approach is focused on determining message types on structural similarities. Despite the cluster refinements we propose to overcome this, a protocol that defines distinct but structurally very similar message types can still lead to clusters of mixed message types. In this case, clusters that contain messages of multiple types mislead the cluster merging heuristic to merge even more misclassified messages into one mixed cluster. In our set of test traces this was the case for DHCP and SMB. As we have shown, even in this case, NEMETYL is superior to previous alignment-based approaches like Netzob.

VI. Future Work

Besides the three evaluated segmenters, other methods of determining atomic chunks of network messages could be devised. However, known methods that may be suited for this task originate from natural language processing [25] and require adaption to be applicable to binary protocols.

Another line of research is to confirm that our approach can be applied to other protocols without significant parameter tuning. To this end, we plan to validate the robustness of NEMETYL with more known and unknown protocols.

VII. Conclusion

In this paper, we presented a novel method of message type identification for unknown network protocols.

NEMETYL aligns messages without relying on identical byte values to determine the similarity of field candidates or single bytes. With this method we are able to efficiently identify message types of binary protocols. The novelty of our approach is that we abstract from discrete byte values to feature vectors that allow for a similarity measure with a continuous value range. Thus, we are able to discover

structural patterns, which remain hidden when only exact value matches are considered. We use message segments, not bytes, as atomic parts of a message, and apply sequence alignment to it. We combine Hirschberg alignment with DBSCAN as cluster algorithm to improve performance over agglomerative clustering. This results in another benefit of the approach of NEMETYL over previous approaches: It does not require to select any parameter a-priori for the analysis of an unknown protocol. We accomplish this by proposing methods to automatically configure all employed algorithms, particularly including DBSCAN clustering.

We evaluated our approach to validate different aspects of our solution. To have enough information about the protocol specifications of our test traces for deriving the reverse engineering quality, we used known protocols for a quantitative comparison. The results for three different segmentation algorithms show that NEMETYL has considerable advantages in message type identification result quality over previous approaches. Since using a similarity measure with a continuous value range for message parts and messages to analyze an unknown protocol, our approach denotes a fundamentally new method for analyzing binary protocols.

Acknowledgements

The authors would like to thank David Mödinger for his assistance in clarifying the notation of this paper.

References

[1] J. Narayan, S. K. Shukla, and T. C. Clancy, "A Survey of Automatic Protocol Reverse Engineering Tools," *ACM Computing Surveys*, vol. 48, no. 3, Dec. 2015, ACM.

[2] J. Duchêne, C. L. Guernic, E. Alata, V. Nicomette, and M. Kaâniche, "State of the Art of Network Protocol Reverse Engineering Tools," *Journal of Computer Virology and Hacking Techniques*, vol. 14, no. 1, Feb. 2018, Springer.

[3] S. Kleber, L. Maile, and F. Kargl, "Survey of Protocol Reverse Engineering Algorithms: Decomposition of Tools for Static Traffic Analysis," *IEEE Communications Surveys and Tutorials*, vol. 21, no. 1, Feb. 2019, Firstquarter.

[4] CAPEC Content Team, *CAPEC - CAPEC-192: Protocol Reverse Engineering (Version 2.6)*, Jun. 2014. [Online]. Available: https : / / web . archive . org / web / 20140725160124 / http : / / capec . mitre . org : 80 / data / definitions/192.html (visited on 08/22/2018).

[5] C. Leita, K. Mermoud, and M. Dacier, "ScriptGen: An Automated Script Generation Tool for Honeyd," in *Proceedings of the 21st Annual Computer Security Applications Conference*, IEEE, 2005.

[6] T. Krueger, H. Gascon, N. Krämer, and K. Rieck, "Learning Stateful Models for Network Honeypots," in *Proceedings of the 5th ACM Workshop on Security and Artificial Intelligence*, ACM, 2012.

[7] H. Gascon, C. Wressnegger, F. Yamaguchi, D. Arp, and K. Rieck, "PULSAR: Stateful Black-Box Fuzzing of Proprietary Network Protocols," in *11th International Conference of Security and Privacy in Communication Networks, Revised Selected Papers*, Springer, 2015.

[8] C. Wressnegger, A. Kellner, and K. Rieck, "ZOE: Content-based Anomaly Detection for Industrial Control Systems," in *Proceedings of the 48th Conference on Dependable Systems and Networks*, 2018.

[9] J. Antunes, N. Neves, and P. Verissimo, "Reverse Engineering of Protocols from Network Traces," in *18th Working Conference on Reverse Engineering*, IEEE, 2011.

[10] G. Bossert, F. Guihéry, and G. Hiet, "Towards Automated Protocol Reverse Engineering Using Semantic Information," in *Proceedings of the 9th ACM Symposium on Information, Computer and Communications Security*, ACM, 2014.

[11] Y.-H. Goo, K.-S. Shim, M.-S. Lee, and M.-S. Kim, "Protocol Specification Extraction Based on Contiguous Sequential Pattern Algorithm," *IEEE Access*, vol. 7, pp. 36 057–36 074, 2019.

[12] S. B. Needleman and C. D. Wunsch, "A General Method Applicable to the Search for Similarities in the Amino Acid Sequence of Two Proteins," *Journal of Molecular Biology*, vol. 48, no. 3, pp. 443–453, Mar. 1970, Elsevier.

[13] L. Wang and T. Jiang, "On the Complexity of Multiple Sequence Alignment," *Journal of Computational Biology*, vol. 1, no. 4, pp. 337–348, Jan. 1994, Mary Ann Liebert, Inc.

[14] R. R. Sokal and C. D. Michener, "A Statistical Method for Evaluating Systematic Relationships," *University of Kansas Scientific Bulletin*, vol. XXXVIII-2, no. 22, pp. 1409–1438, Mar. 1958, University of Kansas.

[15] S. Kleber, H. Kopp, and F. Kargl, "NEMESYS: Network Message Syntax Reverse Engineering by Analysis of the Intrinsic Structure of Individual Messages," in *12th USENIX Workshop on Offensive Technologies*, USENIX Association, 2018.

[16] M. A. Beddoe, "Network Protocol Analysis using Bioinformatics Algorithms," McAfee Inc., Tech. Rep., 2004.

[17] W. Cui, J. Kannan, and H. J. Wang, "Discoverer: Automatic Protocol Reverse Engineering from Network Traces," in *Proceedings of 16th USENIX Security Symposium*, USENIX Association, 2007.

[18] I. Bermudez, A. Tongaonkar, M. Iliofotou, M. Mellia, and M. M. Munafò, "Towards Automatic Protocol Field Inference," *Computer Communications*, vol. 84, Jun. 2016, Elsevier.

[19] G. N. Lance and W. T. Williams, "Computer Programs for Hierarchical Polythetic Classification ("Similarity Analyses")," *The Computer Journal*, vol. 9, no. 1, pp. 60–64, May 1966.

[20] T. F. Smith, M. S. Waterman, and W. M. Fitch, "Comparative biosequence metrics," *Journal of Molecular Evolution*, vol. 18, no. 1, pp. 38–46, Jan. 1981.

[21] M. Ester, H.-P. Kriegel, J. Sander, and X. Xu, "A Density-Based Algorithm for Discovering Clusters in Large Spatial Databases with Noise," in *Proceedings of 2nd International Conference on Knowledge Discovery and Data Mining*, AAAI Press, 1996.

[22] D. S. Hirschberg, "A linear space algorithm for computing maximal common subsequences," *Communications of the ACM*, vol. 18, no. 6, pp. 341–343, Jun. 1975.

[23] C. D. Manning, P. Raghavan, and H. Schütze, "Introduction to Information Retrieval," p. 569, 2009.

[24] H. Fereidooni, J. Classen, T. Spink, P. Patras, M. Miettinen, A.-R. Sadeghi, M. Hollick, and M. Conti, "Breaking Fitness Records Without Moving: Reverse Engineering and Spoofing Fitbit," in *20th International Symposium Research in Attacks, Intrusions, and Defenses*, Springer, 2017.

[25] Y. Wang, X.-C. Yun, M. Z. Shafiq, L. Wang, A. X. Liu, Z. Zhang, D. Yao, Y. Zhang, and L. Guo, "A Semantics Aware Approach to Automated Reverse Engineering Unknown Protocols," in *20th IEEE International Conference on Network Protocols*, IEEE, 2012.