# NetPlier: Probabilistic Network Protocol Reverse Engineering from Message Traces

Yapeng Ye, Zhuo Zhang, Fei Wang, Xiangyu Zhang, Dongyan Xu
Department of Computer Science, Purdue University
{ye203, zhan3299, feiwang, xyzhang, dxu}@purdue.edu

*Abstract*—**Network protocol reverse engineering is an important challenge with many security applications. A popular kind of method leverages network message traces. These methods rely on pair-wise sequence alignment and/or tokenization. They have various limitations such as difficulties of handling a large number of messages and dealing with inherent uncertainty. In this paper, we propose a novel probabilistic method for network trace based protocol reverse engineering. It first makes use of multiple sequence alignment to align all messages and then reduces the problem to identifying the keyword field from the set of aligned fields. The keyword field determines the type of a message. The identification is probabilistic, using random variables to indicate the likelihood of each field (being the true keyword). A joint distribution is constructed among the random variables and the observations of the messages. Probabilistic inference is then performed to determine the most likely keyword field, which allows messages to be properly clustered by their true types and enables the recovery of message format and state machine. Our evaluation on 10 protocols shows that our technique substantially outperforms the state-of-the-art and our case studies show the unique advantages of our technique in IoT protocol reverse engineering and malware analysis.**

## I. Introduction

Network protocol reverse engineering is an important challenge to cyber-security. Many applications that are of interest for security analysts often have their own undocumented communication protocols. For example, autonomous vehicles utilize CAN bus and FlexRay, control systems use Modbus and DNP3, online chatting/conferencing applications have their customized protocols. Many security analysis such as static/symbolic vulnerability scanning [40], [24], exploit generation [79], [19], fuzzing [65], [43], [44], [31], attack detection [15], [29], and malware behavior analysis [75], [18] require precise modeling of the network protocol. For instance, knowing the protocol of a networking application is critical to seed input generation in fuzzing; malware analysis often requires composing well-formed messages to the *Command and Control* (C&C) server so that hidden behaviors can be triggered by the appropriate server responses [23], [83]; and static/symbolic analysis needs to properly model networking functions otherwise a lot of false positives may be generated.

Existing protocol reverse engineering techniques fall into a few categories. The first category leverages program analysis [28], [57], [82], [33], [59], [32]. By analyzing the rich semantics of the application implementation (e.g., how input buffer is accessed), these techniques may achieve high accuracy in reverse engineering. However, most of these techniques require access to program binaries, which is often infeasible in practice. For example, some IoT firmware is not accessible due to their protection mechanism; it is hard to conduct dynamic analysis if the binaries are packed or obfuscated. Even if the binaries for a client application were available, its counterpart on the server side would be much more difficult to acquire. Therefore, the other category focuses on using network traces, which could be acquired by eavesdropping on the network. There are two main techniques for network trace based reverse engineering: *alignment based* (e.g., PIP [22], ScritGen [55], and Netzob [26]) and *token based* (e.g., Veritas [81] and Discoverer [35]). The former leverages various sequence alignment algorithms to align message pairs and compute similarity scores. Messages are clustered based on such scores. Formats are then derived by analyzing the commonality of messages within clusters. However, the diversity of message contents substantially degrades the quality of alignment, causing problems for downstream analysis. Token based methods propose to first tokenize the messages (e.g., to textual fields and binary fields) before alignment to reduce variations. However, these techniques often require delimiters to identify tokens (which may not exist for binary protocols) or generate excessive clusters as tokenization is based on deterministic heuristics. That is, ad-hoc rules are used to perform tokenization and these rules may not hold in many cases. Existing techniques do not model such uncertainty and hence often yield incorrect results. More discussion of such limitations can be found in Section II.

We observe that the key to network protocol reverse engineering is to identify the *keyword* field that determines the type of a message. While there are many heuristics to help locating such keywords, these heuristics are largely uncertain. The reverse engineering of both the client side and the server side can be coupled to achieve synergy because they have strong correspondences. Based on these observations, we propose a novel probabilistic approach to reverse engineering network protocols. Our technique is completely network trace based and does not require access to source code or binary code. Specifically, it leverages *multiple sequence alignment* (MSA) [39] that is widely used in biometrics to avoid the expensive pair-wise alignment in existing work. The alignment is conservative and initially performed on all the messages. As such, the common structure shared by all messages can be disclosed and such structure ought to include the keyword field as the parser needs to parse the keyword field before it can perform type-specific

Network and Distributed Systems Security (NDSS) Symposium 2021
21-24 February 2021
ISBN 1-891562-66-5
https://dx.doi.org/10.14722/ndss.2021.24531
www.ndss-symposium.org

parsing. After the alignment, a probabilistic method is used to determine which (aligned) field is the keyword. To model the inherent uncertainty, we introduce a random boolean variable that predicates if a field is the keyword. We speculatively classify all the messages based on the values of the tentative keyword. Observations can be made from the clustering results, such as if messages within a cluster have similarity and if the corresponding messages from the client side and the server side fall into corresponding clusters. Random variables are introduced to denote the confidence of these observations. A joint probability distribution is constituted by considering the correlations between the keyword variable and observation variables. Posterior marginal probabilities can be computed for keyword variables to indicate the likelihood of individual fields being the true keyword. Once the keyword is identified, messages are clustered based on the keyword values and type specific structures can be disclosed by aligning and analyzing messages in clusters.

Our contributions are summarized as follows.

- We address a key challenge in network protocol reverse engineering – keyword identification, which allows correctly clustering network messages and enables high precision in downstream analysis such as field identification and state machine reconstruction.
- We formulate keyword identification as a probabilistic inference problem, which allows us to naturally model the inherent uncertainty.
- We build an end-to-end system NETPLIER, which stands for "*Probabilistic* NET*work* P*rotoco*L *Reverse Eng*I*ne*ER*ing*". It takes network traces as input and produces the final message format.
- We evaluate NETPLIER on 10 protocols commonly used in competitor projects. Our results show that NETPLIER can achieve 100% homogeneity and 97.9% completeness, whereas the state-of-the-art techniques can only achieve around 92% homogeneity and 52.3% completeness. To validate generality, we use NETPLIER to reverse engineer wireless physical-layer protocols and multiple unknown protocols used in real IoT devices. We also perform two case studies: (i) reverse engineering the protocol for Google Nest, a real world IoT smart app, allowing us to manipulate the A/C unit controlled by the app, and (ii) reserve engineering the C&C protocol for a recent malware, allowing us to expose its hidden malicious behaviors. NETPLIER and data are publicly available at [6].

## II. MOTIVATION

In this section, we use an example to illustrate the limitations of existing network trace based protocol reverse engineering methods and motivate our technique.

### A. Motivation Example

The trace snippet in Figure 1 contains a sequence of *messages* of Distributed Network Protocol 3 (DNP3), which is a communication protocol used in industrial control systems. The trace records information about sent time, IP addresses and ports of the source and destination, and data for each message. The message data includes contents of protocols ranging from application layer to physical layer. Each protocol's message data is composed of several *fields*. Consider the message data of DNP3 in Figure 1. The bytes in bold are a specific field denoting message type. It is also called the *keyword*. Each message type has its own format, which defines the syntax of this type. The sending and receiving of messages are stateful within a network session. State transitions are usually described by a state machine. To be specific, when a client or server receives a new message, it determines its message type by the keyword, parses the remaining fields following the format of this type, and then takes actions according to the state machine. For example in Figure 1, there are four communication connections, which start with an *Unsolicited Response* message $m_{c_0}$, $m_{c_2}$, $m_{c_3}$, and $m_{c_4}$ from the client and a corresponding *Confirm* message $m_{s_0}$, $m_{s_2}$, $m_{s_3}$, and $m_{s_4}$ from the server, respectively. After a connection is established, the server could make requests with different commands, e.g., to *Write* like $m_{s_1}$, $m_{s_5}$, and $m_{s_6}$ or to *Read* like $m_{s_7}$, and the client would confirm with the *Response* messages (e.g., $m_{c_1}$, $m_{c_5}$, $m_{c_6}$, and $m_{c_7}$).

The main goal of protocol reverse engineering is to infer a protocol's syntax and semantics. The first step of protocol reverse engineering is to group messages of the same type into a *cluster*. Clustering is a crucial step as its results determine the accuracy of further format and state machine inference. Existing works usually consider messages from different directions separately. In the following, we use messages from the client as an example ($m_{c_0} - m_{c_7}$) and discuss how existing techniques and our technique conduct clustering. The ideal clustering result is to put messages $m_{c_0}, m_{c_2}, m_{c_3}, m_{c_4}$ into a cluster, and messages $m_{c_1}, m_{c_5}, m_{c_6}$, and $m_{c_7}$. into another cluster.

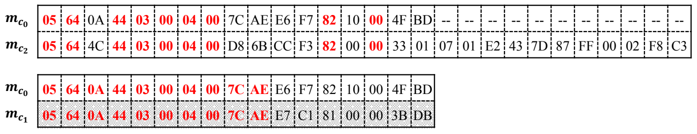
### B. Alignment-based Clustering

Sequence alignment algorithms, such as Needleman & Wunsch [64], are originally used in Biology for the purpose of arranging DNA, RNA, and protein sequences to identify regions of similarity. This idea was borrowed by a large body of existing network trace based protocol reverse engineering methods, such as PIP [22], ScriptGen [55], and Netzob [26]. They use pairwise sequence alignment algorithms to align each pair of messages and compute a similarity score by the alignment results. After constructing a similarity matrix, the messages/clusters with the highest similarity are recursively merged by a clustering algorithm, such as UPGMA [74]. Protocol format and state machine are then derived from the clustering results.

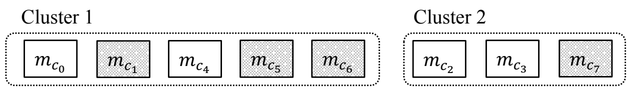
The alignment-based clustering methods work on an assumption that messages are of the same type if they have similar sequences of values. However, this assumption is not true all the time. For messages of the same type, they may have different values for same fields. For messages of different types, they may share some common fields and have the same values. Figure 2a shows the alignment results of message pair $\langle m_{c_0}, m_{c_2}\rangle$ and $\langle m_{c_0}, m_{c_1}\rangle$. The red bytes are the same value aligned together. We can see that although $m_{c_0}$ and $m_{c_2}$ are of the same type, their similarity is lower than $m_{c_0}$ and $m_{c_1}$, which are of different types (illustrated by the shade). Based on this weak assumption, the clustering results are problematic.

| ID | Time (s) | SRC IP : Port | DST IP : Port | Data | Type |
|---|---|---|---|---|---|
| $m_{c_0}$ | 0.00 | 10.0.0.3:20000 | 10.0.0.8:2789 | 05 64 0A 44 03 00 04 00 7C AE E6 F7 **82** 10 00 4F BD | Unsolicited Response |
| $m_{s_0}$ | 3.04 | 10.0.0.8:2789 | 10.0.0.3:20000 | 05 64 08 C4 04 00 03 00 B4 B8 C0 D7 **00** 7A CE | Confirm |
| $m_{s_1}$ | 3.04 | 10.0.0.8:2789 | 10.0.0.3:20000 | 05 64 12 C4 04 00 03 00 1E 7C C1 C1 **02** 32 01 07 01 EB E4 5A 87 FF 00 28 01 | Write |
| $m_{c_1}$ | 3.06 | 10.0.0.3:20000 | 10.0.0.8:2789 | 05 64 0A 44 03 00 04 00 7C AE E7 C1 **81** 00 00 3B DB | Response |
| $m_{c_2}$ | 2256.60 | 10.0.0.3:20000 | 10.0.0.8:2828 | 05 64 4C 44 03 00 04 00 D8 6B CC F3 **82** 00 00 33 01 07 01 E2 43 7D 87 FF 00 02 F8 C3 | Unsolicited Response |
| $m_{s_2}$ | 2256.66 | 10.0.0.8:2828 | 10.0.0.3:20000 | 05 64 08 C4 04 00 03 00 B4 B8 C3 D3 **00** 78 D3 | Confirm |
| $m_{c_3}$ | 2258.06 | 10.0.0.3:20000 | 10.0.0.8:2828 | 05 64 47 44 03 00 04 00 54 62 CD F4 **82** 00 00 33 01 07 01 D5 47 7D 87 FF 00 02 49 5C | Unsolicited Response |
| $m_{s_3}$ | 2258.07 | 10.0.0.8:2828 | 10.0.0.3:20000 | 05 64 08 C4 04 00 03 00 B4 B8 C4 D4 **00** 31 18 | Confirm |
| $m_{c_4}$ | 5847.38 | 10.0.0.3:20000 | 10.0.0.8:1086 | 05 64 0A 44 03 00 04 00 7C AE C0 F0 **82** 90 00 43 A2 | Unsolicited Response |
| $m_{s_4}$ | 5850.02 | 10.0.0.8:1086 | 10.0.0.3:20000 | 05 64 08 C4 04 00 03 00 B4 B8 C0 D0 **00** 1B 49 | Confirm |
| $m_{s_5}$ | 5850.57 | 10.0.0.8:1086 | 10.0.0.3:20000 | 05 64 12 C4 04 00 03 00 1E 7C C4 C4 **02** 32 01 07 01 5B 1E B4 87 FF 00 DA 67 | Write |
| $m_{c_5}$ | 5850.66 | 10.0.0.3:20000 | 10.0.0.8:1086 | 05 64 0A 44 03 00 04 00 7C AE C4 C4 **81** 80 00 80 A3 | Response |
| $m_{s_6}$ | 5850.91 | 10.0.0.8:1086 | 10.0.0.3:20000 | 05 64 0E C4 04 00 03 00 6D D3 C6 C6 **02** 50 01 00 07 07 00 34 61 | Write |
| $m_{c_6}$ | 5850.98 | 10.0.0.3:20000 | 10.0.0.8:1086 | 05 64 0A 44 03 00 04 00 7C AE C6 C6 **81** 00 00 0A 36 | Response |
| $m_{s_7}$ | 5851.20 | 10.0.0.8:1086 | 10.0.0.3:20000 | 05 64 14 C4 04 00 03 00 C7 17 C7 C7 **01** 3C 02 06 3C 03 06 3C 04 06 3C 01 06 6B AE | Read |
| $m_{c_7}$ | 5851.29 | 10.0.0.3:20000 | 10.0.0.8:1086 | 05 64 4E 44 03 00 04 00 6F 4D C7 C7 **81** 00 00 01 01 00 00 05 19 0A 02 00 00 05 C3 47 | Response |

Fig. 1: Motivation example: establishing multiple DNP3 (an industrial control protocol) connections and performing some data transfer; plain and shaded messages originate from the client and server side, respectively.


$m_{c_0}$ 05 64 0A 44 03 00 04 00 7C AE E6 F7 82 10 00 4F BD -- -- -- -- -- -- -- -- -- -- --
$m_{c_2}$ 05 64 4C 44 03 00 04 00 D8 6B CC F3 82 00 00 33 01 07 01 E2 43 7D 87 FF 00 02 F8 C3
$m_{c_0}$ 05 64 0A 44 03 00 04 00 7C AE E6 F7 82 10 00 4F BD
$m_{c_1}$ 05 64 0A 44 03 00 04 00 7C AE E7 C1 81 00 00 3B DB

(a) Pairwise sequence alignment example



(b) Clustering results

Fig. 2: Clustering by Netzob. Plain-text and shaded messages belong to two respective types.

Figure 2b shows the clustering results by Netzob. It generates two clusters and both contain messages of different types. Based on the wrong clustering results, the further format and state machine inference will also be inaccurate.

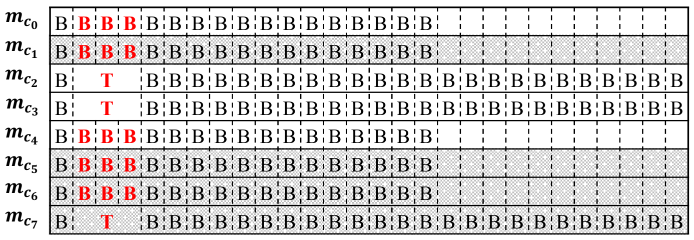
Another limitation of alignment-based clustering methods is that it requires a threshold of similarity score to decide which clusters should be merged together in the recursive clustering step when using algorithms such as UPGMA. The clustering results are sensitive to this threshold and different protocols should use different thresholds. However, when reverse engineering an unknown protocol, it is hard to compute the optimal threshold without the ground truth. Normally, we can only use a general threshold trained from other well studied protocols. As such, the clustering accuracy likely degenerates.

### *C. Token-based Clustering*

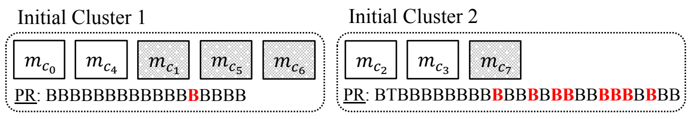
Token-based clustering methods split a message into tokens and then group messages by specific token values or token types. Most methods in this line, such as ASAP [52], Veritas [81], Prisma [51], and ProDecoder [80], rely on predefined delimiters or n-grams to split messages into tokens, and then search for the ones with the most frequent values which can be further used to cluster messages.

Another token-based clustering strategy is to use token type patterns. Discoverer [35], the state-of-the-art token based method, uses token type patterns to conduct initial clustering, followed by a combination of representative token values and sequence alignment algorithms to improve clustering results. Figure 3a shows the tokenization results by Discoverer. It considers consecutive bytes with printable ASCII values as a text token, leveraging the observation that the same type of network messages have the same mixture of binary sequences and textual strings. So the second to the fourth bytes in $m_{c_2}$, $m_{c_3}$, and $m_{c_7}$ are marked as a text token `T`, and the other individual bytes are marked as binary tokens `B`. After tokenization, it observes two different token patterns, sequence "`BBBB ... B`" for $m_{c_0}$, $m_{c_1}$, $m_{c_4}$, etc. and sequence "`BTBB ... B`" for $m_{c_2}$, $m_{c_3}$, and $m_{c_7}$. The differences of the two patterns are highlighted in red. As such, Discoverer produces two initial clusters as shown in Figure 3b. Then it divides each cluster into sub-clusters by values of *potential representative* (PR). Finally, it utilizes message alignment to merge some of the sub-clusters to a larger cluster to avoid over-partitioning. For example, in the first cluster ($m_{c_0}$, $m_{c_4}$, $m_{c_1}$, $m_{c_5}$, $m_{c_6}$), the token in red ('B') contains only two different values (81 and 82), which could be considered as a representative token and used to obtain new sub-clusters (Cluster 1 and Cluster 2 in Figure 3c).

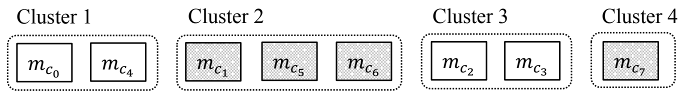
Finally it produces four clusters as shown in Figure 3c. Although there is only one type in each cluster, each ground-truth type (denoted by shade or no-shade) is suboptimally divided into two smaller types (clusters). There are many reasons causing this issue. First, there are no clear delimiters in binary protocols. Hence most bytes are considered as individual tokens, diminishing the value of tokenization as little structural information is exposed. Also, the values of binary tokens sometimes lie in the range of text tokens so that these binary tokens could be mistaken for text tokens (e.g., the text tokens in Figure 3a). A text string shorter than the minimum length (for qualifying as a text token) is also wrongly marked as binary tokens. Another problem is that there could be multiple representative tokens found in the recursive clustering and merging step. All these reasons lead to excessive token types. In our experiments (Section V), Discoverer always suffers from redundant clusters, which indicates its clustering results are not concise.


$m_{c_0}$ B B B B B B B B B B B B B B B B B
$m_{c_1}$ B B B B B B B B B B B B B B B B B
$m_{c_2}$ B T B B B B B B B B B B B B B B B B B B B B B B B B
$m_{c_3}$ B T B B B B B B B B B B B B B B B B B B B B B B B B
$m_{c_4}$ B B B B B B B B B B B B B B B B B
$m_{c_5}$ B B B B B B B B B B B B B B B B B
$m_{c_6}$ B B B B B B B B B B B B B B B B B
$m_{c_7}$ B T B B B B B B B B B B B B B B B B B B B B B B B B

(a) Tokenization

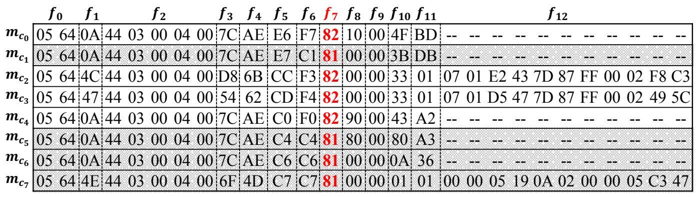

Initial Cluster 1
$m_{c_0}$ $m_{c_4}$ $m_{c_1}$ $m_{c_5}$ $m_{c_6}$
PR: BBBBBBBBBBBBBBBBB
Initial Cluster 2
$m_{c_2}$ $m_{c_3}$ $m_{c_7}$
PR: BTBBBBBBBBBBBBBBBBBBBBBBBB

(b) Initial clustering



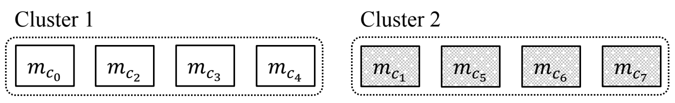
(c) Clustering results

Fig. 3: Clustering by Discoverer



| | $f_0$ | | $f_1$ | $f_2$ | | | | | $f_3$ | $f_4$ | $f_5$ | $f_6$ | $f_7$ | $f_8$ | $f_9$ | $f_{10}$ | $f_{11}$ | $f_{12}$ | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| $m_{c_0}$ | 05 | 64 | 0A | 44 | 03 | 00 | 04 | 00 | 7C | AE | E6 | F7 | **82** | 10 | 00 | 4F | BD | -- | -- | -- | -- | -- | -- | -- | -- | -- | -- | -- |
| $m_{c_1}$ | 05 | 64 | 0A | 44 | 03 | 00 | 04 | 00 | 7C | AE | E7 | C1 | **81** | 00 | 00 | 3B | DB | -- | -- | -- | -- | -- | -- | -- | -- | -- | -- | -- |
| $m_{c_2}$ | 05 | 64 | 4C | 44 | 03 | 00 | 04 | 00 | D8 | 6B | CC | F3 | **82** | 00 | 00 | 33 | 01 | 07 | 01 | E2 | 43 | 7D | 87 | FF | 00 | 02 | F8 | C3 |
| $m_{c_3}$ | 05 | 64 | 47 | 44 | 03 | 00 | 04 | 00 | 54 | 62 | CD | F4 | **82** | 00 | 00 | 33 | 01 | 07 | 01 | D5 | 47 | 7D | 87 | FF | 00 | 02 | 49 | 5C |
| $m_{c_4}$ | 05 | 64 | 0A | 44 | 03 | 00 | 04 | 00 | 7C | AE | C0 | F0 | **82** | 90 | 00 | 43 | A2 | -- | -- | -- | -- | -- | -- | -- | -- | -- | -- | -- |
| $m_{c_5}$ | 05 | 64 | 0A | 44 | 03 | 00 | 04 | 00 | 7C | AE | C4 | C4 | **81** | 80 | 00 | 80 | A3 | -- | -- | -- | -- | -- | -- | -- | -- | -- | -- | -- |
| $m_{c_6}$ | 05 | 64 | 0A | 44 | 03 | 00 | 04 | 00 | 7C | AE | C6 | C6 | **81** | 00 | 00 | 0A | 36 | -- | -- | -- | -- | -- | -- | -- | -- | -- | -- | -- |
| $m_{c_7}$ | 05 | 64 | 4E | 44 | 03 | 00 | 04 | 00 | 6F | 4D | C7 | C7 | **81** | 00 | 00 | 01 | 01 | 00 | 00 | 05 | 19 | 0A | 02 | 00 | 00 | 05 | C3 | 47 |

(a) Multiple sequence alignment and keyword inference


Cluster 1
$m_{c_0}$ $m_{c_2}$ $m_{c_3}$ $m_{c_4}$
Cluster 2
$m_{c_1}$ $m_{c_5}$ $m_{c_6}$ $m_{c_7}$

(b) Clustering results

Fig. 4: Clustering by NETPLIER

### D. Our Technique

**Insights.** From the above discussion, we observe that both alignment-based clustering and token-based clustering rely on the assumption that messages are of the same type if they have similar values or patterns. However, in many cases this assumption does not hold and incurs inaccurate clustering. *In fact when a client or a server receives a message, it determines the message type only by the keyword.* Thus, if we can infer the field denoting the keyword, we would obtain the ideal clustering results. Note that although some token-based clustering methods use representative tokens for clustering [35], [80], they only search for such tokens by statistics such as frequency, which usually generates more than one representative token and then leads to redundant clustering.

Another insight is to take better advantage of network traces, which are the only input for trace based methods. Existing works only analyze message data from one side (the client side or the server side) to study the aforementioned hints. However, *we could observe more hints if we consider the message traces from both sides, especially their correspondence*. For example, in Figure 1 we can see that all the *Unsolicited Response* messages $m_{c_0}$, $m_{c_2}$, $m_{c_3}$, and $m_{c_4}$ from the client side have the *Confirm* messages $m_{s_0}$, $m_{s_2}$, $m_{s_3}$, and $m_{s_4}$ from the server as the response (for setting up a new connection). Also, the *Write* messages $m_{s_1}$, $m_{s_5}$, and $m_{s_6}$ sent by the server always trigger *Response* messages, i.e., $m_{c_1}$, $m_{c_5}$, and $m_{c_6}$ from the client. These additional hints could be used to improve and validate clustering results.

As we already know that all these hints have inherent uncertainty as arbitrary byte sequences could appear as hints, the results may be incorrect/contradictory if we only consider few hints for clustering. For example, alignment-based clustering methods only use the hint that messages with high similarity are of the same type. Inspired by the application of *probabilistic inference* in specification extraction [60], [34] and program analysis [84], *a more reasonable solution is to combine various kinds of hints together in a probabilistic fashion*. Specifically, a prior probability is assigned to each hint denoting its uncertainty instead of making a simple deterministic call. Probabilistic inference aggregates these hints and computes a posterior distribution from which we can derive the *most likely* keywords and clustering.

**Our Idea.** We use *multiple sequence alignment* (MSA) algorithms on messages from both the client and server sides and partition messages into a list of fields. MSA tends to be conservative and only produces a comprehensive list of fields, which provides a solid starting point. For each field, we introduce a random variable to denote the probability of being the keyword. Assume a field is the keyword, messages could be grouped into different clusters by the value of the field, and these clusters would satisfy some constraints, e.g., message similarity constraints, remote coupling constraints, structure coherence constraints, and dimension constraints. For each constraint, We compute probabilities to serve as the degree of compliance that we observe. With these probabilities, we then perform probabilistic inference to derive the posterior probability of random variables that denote our assumption, i.e., the current field is the keyword. After checking all fields, we can pick the one with the highest probability as the keyword, and use it to cluster messages. In the motivation example, we generate 12 fields from the MSA results of client messages, as shown in Figure 4a. After probabilistic inference, field $f_7$ is chosen as the keyword with the highest posterior probability. Then we can generate two correct clusters by the values of $f_7$, which is show in Figure 4b.

## III. SYSTEM DESIGN

In this section, we discuss the system design, including preprocessing, keyword field candidate generation, probabilistic keyword identification, iterative alignment and clustering, and format and state machine inference. Figure 5 shows the workflow of NETPLIER.

### A. Preprocessing

The input of NETPLIER is network traces which could be captured by packet analyzers such as *tcpdump*. The packets in traces follow the network layer models. The unknown protocols we aim to reverse engineer are usually in the application layer. Based on the knowledge of other existing protocols,

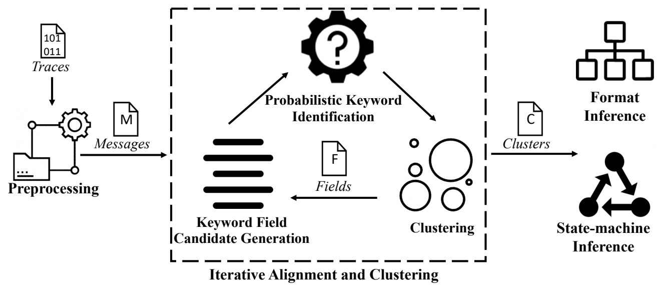
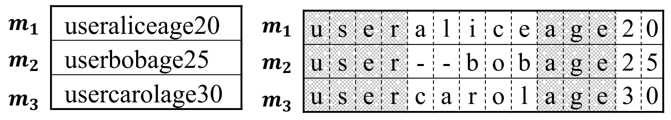

Traces
Messages
Preprocessing
Probabilistic Keyword Identification
Fields
Keyword Field Candidate Generation
Clustering
Iterative Alignment and Clustering
Clusters
Format Inference
State-machine Inference

Fig. 5: System design


$m_1$ useraliceage20
$m_2$ userbobage25
$m_3$ usercarolage30
$m_1$ u s e r a l i c e a g e 2 0
$m_2$ u s e r - - b o b a g e 2 5
$m_3$ u s e r c a r o l a g e 3 0

(a) Original messages (b) Alignment results

Fig. 6: Examples with variable-length fields

we can reconstruct messages in these protocols, extract useful information (e.g., port numbers from the network layer), and discard data in irrelevant protocols. Finally, we standardize network messages to include the following information: timestamp, IP address(es), port number(s), and data of the target protocol. An example of such standardized messages can be found in Figure 1.

By timestamp, IP address, and port number, we can group messages into communication sessions. For example, there are three sessions in the example shown in Figure 1, where $m_{c_0}, m_{s_0}, m_{s_1}, m_{c_1}$ belong to the first session, $m_{c_2}, m_{s_2}, m_{c_3}, m_{s_3}$ belong to the second session, and the other messages belong to the third session. The session information will be used in the probabilistic inference and state machine inference stages.

## B. Keyword Field Candidate Generation

As mentioned earlier, identifying keywords (in network messages) is critical. In this stage, we identify a set of fields that are candidates for keywords.

Message data is composed of multiple fields. For the example in Figure 1, all messages have similar field structures and these fields are of the same length (except the last field). Hence we can easily acquire the value of a field by its position. However, for complex protocols, messages may have different structures and some fields may have a variable length, which makes a field appear at different positions in different messages. For example, messages in Figure 6a have a field for user name, which has a variable length. Intuitively, the idea of recognizing such fields is to identify the fixed length fields that bound fields of a variable length, by message alignment. We observe that messages tend to share some common values, especially for fixed length fields, e.g., "user" and "age" in Figure 6a. Hence we can align messages to expose such common sequences across messages, and then identify the variable-length field(s) in between them. If multiple consecutive variable-length fields are present in between two bounding fixed-length fields, NETPLIER may recognize these variable-length fields as one monolithic variable-length field. In practice, we rarely see such cases. Note that this is a generally hard problem for any trace based revere engineering techniques to precisely separate them.

As discussed earlier, pairwise alignment algorithms are widely used by existing methods. However, pairwise alignment only compares two sequences at one time, which substantially affects scalability when the number of samples is large. Therefore, we leverage *multiple sequence alignment* [39], which is an extension of pairwise alignment in Bioinformatics and could align all sequences at a time. There are various strategies used to reduce computational complexity and improve accuracy for multiple sequence alignment. Here, we use a combination of progressive methods [39] and iterative refinement [62]. Progressive methods align the most similar sequences first and then progressively add other sequences to the alignment results. Iterative refinement methods iteratively realign sequence subsets of initial global alignment results to improve the accuracy. Figure 6b shows the result after multiple sequence alignment. Gaps (i.e., '-') are inserted into the variable-length fields in order to demonstrate alignment results.

Based on the initial alignment results (on all messages), we partition message data into fields. For text data, we can use predefined delimiters, such as a space character, to partition message data into fields. However, binary data do not have specific delimiters and its fields are usually a few bytes long. We need to use the alignment results in a very conservative way such that it considers every possible candidate of a field. First, we consider each (aligned) byte as a single *unit field*. A unit field is marked as *static* if all message data have the same value for the field, otherwise *dynamic*. Then consecutive static unit fields are merged to a larger unit field. For example, in Figure 4a, fields $f_0$, $f_2$, and $f_9$ are static unit fields, and the others are dynamic (this may not be true as we only show a short snippet with some fields elided due to the space limitation). These unit fields denote a conservative list of candidates for real fields, meaning that *a field in the real specification is a unit field or a concatenation of multiple unit fields*. At the end of this stage, we generate a list that includes all the unit fields and their compositions that are shorter than a threshold (i.e., 10 bytes in this paper). The compositions are also called *compound fields*. The list denotes candidates for keyword fields and is subject to the downstream probabilistic analysis. We bound the size in order to reduce the number of candidate fields to analyze. Note that in $f_{12}$ we combine a sequence of bytes as it is empty for some messages, which means it could not be the keyword field and could be ignored. Although most protocols use similar formats for both client side and server side, some protocols may have substantially different field structures. We hence generate fields for client side and server side separately (while considering their correspondence in probabilistic analysis). Note that although field candidate generation is not complex, it ought to be conservative and include the real (keyword) fields. NETPLIER relies on the later probabilistic analysis to recognize the keyword fields with high accuracy, which in turn allows identifying the other fields and pruning the bogus ones.

## C. *Probabilistic Keyword Identification*

Given a list of keyword candidate fields for both sides, we use a probabilistic method to infer which fields are most likely the keywords. With keyword fields identified, messages of the same type (i.e., having the same keyword value) can be identified and further alignment and analysis can be performed on these messages.

Let fields $f_c$ and $f_s$ be the potential keywords from the client and the server sides, respectively. Client-side messages are speculatively grouped into clusters $(t_{c_0}, t_{c_1}, \dots)$ by $f_c$ and server-side messages are grouped to $(t_{s_0}, t_{s_1}, \dots)$ by $f_s$. In the example shown in Figure 1, the list of candidate fields for client side messages are shown in Figure 4a. The server side messages have a very similar list. Figure 7a shows the clustering results of considering $f_1$ the keyword for messages on both the client and the server sides and Figure 7b shows the results of considering $f_7$ the keyword. For example, with $f_1$ the keyword, $m_{c_0}$, $m_{c_1}$, $m_{c_4}$, $m_{c_5}$, and $m_{c_6}$ belong to a cluster as their $f_1$ fields all have value `A0`, whereas $m_{s_0}$, $m_{s_2}$, $m_{s_3}$, and $m_{s_4}$ belong to a cluster as their $f_1$ values are `08` (see traces in Figure 1).

If the keyword speculation is true, i.e., the messages in a cluster (grouped by the keyword values) are indeed of the same type, we should have the following observations from the generated clusters.

**Observation 1.** Messages in the same cluster should be more similar than messages in different clusters.

**Observation 2.** Clusters on the client side and the server side should have correspondence. In other words, messages belonging to a cluster on one side (e.g., requests from the client side) very likely have their counterparts on the other side (e.g., corresponding responses from the server side) in a cluster too.

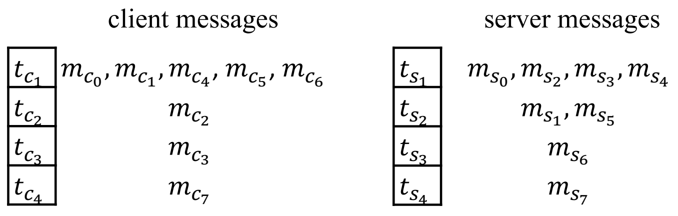
**Observation 3.** Messages in the same cluster follow the same field structure.

**Observation 4.** There should not be too many clusters. In each cluster, there should be enough number of messages.

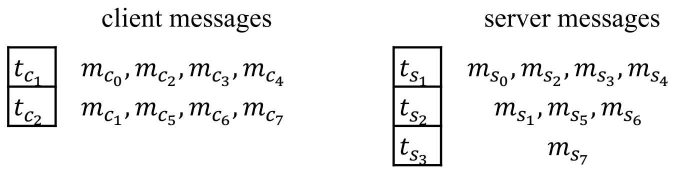
These observations may have uncertainty. In other words, true clusters may not demonstrate such observations and their presence does not necessarily imply true clustering either. Therefore, we introduce a random variable (with boolean value) to indicate if a candidate is the true keyword. The variables (for all the candidates from both client and server) and the observations form a joint probability distribution. We hence formulate keyword identification as a probabilistic inference problem computing the marginal posterior probabilities of keyword random variables given the observations. As we will explain in Section IV, the inference rules may be directional (i.e., Bayesian inference [27]) or un-directional (Markov random fields [48]). We leverage a general graph model called *factor graph* that supports both types. After inference, the random variable with the largest posterior probability indicates the most likely keyword pair.

## D. *Iterative Alignment and Clustering*

MSA may not produce the intended alignment in the first place as it is inherently uncertain as well. As a result, the field


client messages
server messages
$t_{c_1}$ $m_{c_0}, m_{c_1}, m_{c_4}, m_{c_5}, m_{c_6}$
$t_{c_2}$ $m_{c_2}$
$t_{c_3}$ $m_{c_3}$
$t_{c_4}$ $m_{c_7}$
$t_{s_1}$ $m_{s_0}, m_{s_2}, m_{s_3}, m_{s_4}$
$t_{s_2}$ $m_{s_1}, m_{s_5}$
$t_{s_3}$ $m_{s_6}$
$t_{s_4}$ $m_{s_7}$

(a) Clustering results of $f_1$


client messages
server messages
$t_{c_1}$ $m_{c_0}, m_{c_2}, m_{c_3}, m_{c_4}$
$t_{c_2}$ $m_{c_1}, m_{c_5}, m_{c_6}, m_{c_7}$
$t_{s_1}$ $m_{s_0}, m_{s_2}, m_{s_3}, m_{s_4}$
$t_{s_2}$ $m_{s_1}, m_{s_5}, m_{s_6}$
$t_{s_3}$ $m_{s_7}$

(b) Clustering results of $f_7$

Fig. 7: Clustering results of different fields

separation may be problematic, rendering erroneous downstream results. We resort to iterative alignment and clustering to address the problem. Intuitively, assume MSA does not align properly and hence the keyword cannot be correctly identified. Nonetheless, the probabilistic inference and clustering are likely to reduce structural divergence of messages within clusters. As such, for each cluster, we perform MSA and the probabilistic keyword identification. We then compare the resulted keywords with the original ones. If the new keywords can lead to better global partitioning of all the messages (evaluated by metrics derived from the aforementioned four observations), we replace the original keywords with the new ones. The process repeats until no better keywords can be identified. As shown in Section V, the strategy is particularly effective for protocols that have substantial message length variation such as DHCP.

## E. *Format and State Machine Inference*

As discussed earlier, each message is split into several aligned fields after multiple sequence alignment (e.g., Figure 4a). After iterative alignment and clustering, the format for each type can be directly recovered by summarizing the fields of all messages in the same cluster. The format includes fields defined with length ($L$), value ($V$), and field type ($S$: static field with a specific value; '$D$': dynamic field with a list of potential values). For example, in Figure 4a, field $f_0$ can be denoted as $S(V =' 0504')$; field $f_7$ can be $D(L = 1, V = ['82', '81'])$, which is a dynamic field with two potential values; and field $f_{12}$ can be $D(L = (0, 11))$, which is a variable-length field or optional field as it is empty for some messages. New messages could be generated based on the formats.

In addition, we make use of an existing technique [25] to infer state machine. The technique works well when message types are properly defined. The basic idea is to derive message type sequences for each session (in the traces) and aggregate such sequences to form a state machine. Details are elided as it is not our contribution.

Note that full format and state machine inference are not the focus of this paper, which are only provided to evaluate clustering results (Section V-C and Section V-D). More precise inference could be generated if prior knowledge is used to detect some common fields first [26], [54], [69], e.g., length field or address field. This is beyond the scope of this paper.

## IV. Probabilistic Keyword Identification

A key step in our technique is to model uncertainty in keyword identification as a joint distribution of observations and a set of random variables, each denoting if a candidate field is the keyword of messages. In this section, we discuss the details of how to model the uncertainty with probabilities and conduct probabilistic inference with a graphical model.

### *A. Random Variables and Probabilistic Constraints*

The first three columns of Table I define the predicates, their symbols, and descriptions. A predicate has a boolean value and is associated with a random variable in our system. In the rest of the paper, we do not distinguish the terms random variable and predicate. Particularly, the keyword predicate $K(f)$ asserts if field $f$ is the keyword field. The other predicates assert the observations. $M(f, c)$ asserts that the messages in a cluster $c$ by keyword $f$ have higher similarity among themselves than with messages in other clusters; $R(f, c)$ asserts that for the messages in $c$, their corresponding messages on the other side should belong to a same cluster; $S(f, c)$ asserts that the messages in $c$ should have similar field structure; and $D(f)$ is a global assertion (i.e., not specific to a cluster), asserting that keyword $f$ does not lead to too many clusters and each cluster shall have sufficient messages.

The last column in Table I presents the set of constraints related to the predicates. Intuitively, they denote the correlations of the random variables, which can be considered as joint distributions of these variables. Each predicate has two kinds of constraints. The first kind is called the *observation constraint* that associates predicates with *prior probabilities*. They are sub-scripted with a single symbol denoting the associated predicate. For example, constraint $C_m$ is the observation constraint for the message similarity predicate $M(f, c)$. Its body $M(f, c) = 1(p_m)$ means the following "*the predicate $M(f, c)$ has the prior probability of $p_m$ to be true*". The other observation constraints are similarly defined. We will explain how the prior probabilities are systematically derived later in this section.

The second kind of constraints is called the *inference constraints*. They are sub-scripted with an implication relation. The implication could proceed in two ways: *from an observation predicate to a keyword predicate* or *from a keyword predicate to an observation predicate*. They are probabilistic, regulated by an *implication probability*. For example, $C_{k\to m}$ : $K(f) \xrightarrow{p_{m\to}} M(f, c)$ in the third row, fourth column of Table I denotes that if $f$ is the keyword, there is $p_{m\to}$ chance that the messages in cluster $c$ (formed using $f$ as the keyword) have higher inner-cluster similarity than inter-cluster similarity. The following constraint $C_{k\leftarrow m}$ represents the opposite direction of reasoning. Intuitively, the two constraints describe the uncertainty of the relations between $K$ and $M$. For example, even if $f$ is the true keyword, it is still possible that messages of the same type do not have high similarity. Theoretically, the uncertainty, denoted by the implication probabilities, e.g., $p_{m\to}$ and $p_{m\leftarrow}$, follow some normal distribution that can be approximated using predefined constants based on domain knowledge. In practice, existing literature of probability inference typically makes use of pre-defined prior probability values derived from domain knowledge [84], [45], [36], [58], [21], [60], [50]. Existing studies also show that inference results are usually not sensitive to these values due to the iterative nature of inference algorithm. We follow the same practice such as using 0.95 for likely and 0.1 for unlikely, and adjust the implication probabilities based on these two values according to the level of uncertainty of individual observations. For example, the implication probability $p_{r\to}$ for the remote coupling constraint $C_{k\to r}$ (from the keyword to the coupling predicate) is 0.9 as there is little uncertainty. That is, the response messages of the same kind of request messages highly likely belong to the same kind. However, along the opposite direction, $p_{r\leftarrow} = 0.8$ denotes that if corresponding messages on the two sides belong to two respective clusters, we cannot be so confident that $f$ is the right keyword, as such perfect coupling could be by chance. The implication probabilities for message similarity are lower than those for remote coupling as they are more uncertain. In NetPlier, probabilities $p_{\to}$ are set to be 0.8 for message similarity constraints and 0.9 for the others. Probabilities $p_{\leftarrow}$ lies in [0.6, 0.8] depending on cluster sizes. In Section V, we validate these implication probabilities in small datasets (100 messages). We notice that our system is not sensitive to these parameters, consistent with the literature.

TABLE I: Predicate/random variable and constraint definition

| Predicate | Symbol | Definition | Related Constraints |
|---|---|---|---|
| Keyword | $K(f)$ | Field $f$ is the keyword. | |
| Message Similarity | $M(f, c)$ | Messages in cluster $c$ have higher inner similarity than inter similarity. | $C_m : M(f, c) = 1\ (p_m)$<br>$C_{k\to m} : K(f) \xrightarrow{p_{m\to}} M(f, c)$<br>$C_{k\leftarrow m} : K(f) \xleftarrow{p_{m\leftarrow}} M(f, c)$ |
| Remote Coupling | $R(f, c)$ | The corresponding messages of those in cluster $c$ belong to a same cluster. | $C_r : R(f, c) = 1\ (p_r)$<br>$C_{k\to r} : K(f) \xrightarrow{p_{r\to}} R(f, c)$<br>$C_{k\leftarrow r} : K(f) \xleftarrow{p_{r\leftarrow}} R(f, c)$ |
| Structure Coherence | $S(f, c)$ | Messages in cluster $c$ have similar field structure. | $C_s : S(f, c) = 1\ (p_s)$<br>$C_{k\to s} : K(f) \xrightarrow{p_{s\to}} S(f, c)$<br>$C_{k\leftarrow s} : K(f) \xleftarrow{p_{s\leftarrow}} S(f, c)$ |
| Dimension | $D(f)$ | There are not an excessive number of clusters and each cluster has enough number of messages. | $C_d : D(f) = 1\ (p_d)$<br>$C_{k\to d} : K(f) \xrightarrow{p_{d\to}} D(f)$<br>$C_{k\leftarrow d} : K(f) \xleftarrow{p_{d\leftarrow}} D(f)$ |

### *B. Determining Prior Observation Probabilities*

In the following, we discuss in details how to compute the prior probabilities for observation constraints $p_m$, $p_r$, $p_s$ and $p_d$. Different from implication probabilities that denote reasoning uncertainty and are largely stable, these probabilities describe observation data and vary a lot with the field $f$ we use to cluster messages.

**Message Similarity Constraints.**

Based on the MSA results, we can compute the similarity score of a pair of aligned messages:

$$s = \frac{\text{Number of identical bytes}}{\text{Sum of total bytes of the two messages}}$$

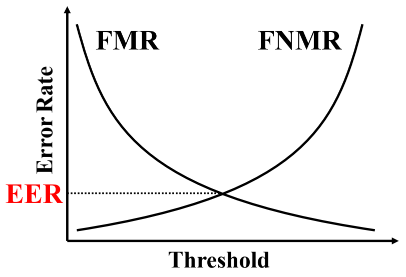


Fig. 8: Example of EER

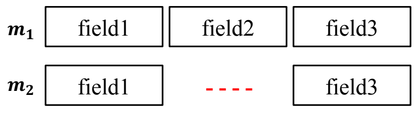


Fig. 9: Example of structure coherence constraints. $m_1$ and $m_2$ belong to different message types with different field structure.

After obtaining similarity scores of all message pairs, a similarity score matrix is constructed. For each keyword candidate field $f$, we can divide all similarity scores into two classes based on its clustering results: inner scores, where the two messages are from the same cluster, and inter scores, where the two messages are from different clusters.

Ideally, message similarity constraints require that all inner scores are higher than inter scores. If so, this constraint would be observed with full confidence, we would hence set $p_m$ to 1. However, the distributions of the two kinds of scores usually overlap, indicating the errors of *false match* and *false non-match*. These terms are drawn from biometrics [68] where multiple sequence alignment is widely used. Intuitively in our context, the former indicates messages of different kinds are undesirably grouped into a cluster, whereas the later indicates messages of the same kind are undesirably placed in different clusters. We quantify the overlap by computing the two errors. Smaller error values lead to a higher prior probability of message similarity constraints.

Specifically, for a threshold $t$ ranging from 0 to 1, we can compute the *False Match Rate* (FMR) and *False Non-Match Rate* (FNMR) as follows.

$$\text{FMR} = \frac{\text{Number of inter scores which are greater than } t}{\text{Number of inter scores}}$$

$$\text{FNMR} = \frac{\text{Number of inner scores which are smaller than } t}{\text{Number of inner scores}}$$

Considering all $t$ in $[0, 1]$, we can draw the curves of FMR and FNMR, as shown in Figure 8. Observe that when $t$ increases, FMR decreases and FNMR increases. To describe the similarity constraints, we need to consider both FMR and FNMR at the same time. Following the practice in biometrics [30], we choose the intersection of the two curves, which balances both FMR and FNMR. The error rate value at the intersection is also called *Equal Error Rate* (EER), which describes the overall accuracy of the clustering results and we have the following.

$$p_m = 1 - EER$$

It means that the lower the EER, the higher confidence we have for the message similarity constraint $M$.

TABLE II: Example of remote coupling constraints. The arrows "$\rightarrow$" and "$\leftarrow$" denote from client to server and server to client, respectively.

| | **Message pairs** | | **Message type pairs of $f_1$** | | **Message type pairs of $f_7$** | |
|---|---|---|---|---|---|---|
| | Traces | Pairs | Traces | Pairs | Traces | Pairs |
| **Session 1** | $m_{c_0} \rightarrow$ | $\langle m_{c_0}, m_{s_0} \rangle$ | $t_{c_1} \rightarrow$ | $\langle t_{c_1}, t_{s_1} \rangle$ | $t_{c_1} \rightarrow$ | $\langle t_{c_1}, t_{s_1} \rangle$ |
| | $\leftarrow m_{s_0}$ | | $\leftarrow t_{s_1}$ | | $\leftarrow t_{s_1}$ | |
| | $\leftarrow m_{s_1}$ | $\langle m_{s_1}, m_{c_1} \rangle$ | $\leftarrow t_{s_2}$ | $\langle t_{s_2}, t_{c_1} \rangle$ | $\leftarrow t_{s_2}$ | $\langle t_{s_2}, t_{c_2} \rangle$ |
| | $m_{c_1} \rightarrow$ | | $t_{c_1} \rightarrow$ | | $t_{c_2} \rightarrow$ | |
| **Session 2** | $m_{c_2} \rightarrow$ | $\langle m_{c_2}, m_{s_2} \rangle$ | $t_{c_2} \rightarrow$ | $\langle t_{c_2}, t_{s_1} \rangle$ | $t_{c_1} \rightarrow$ | $\langle t_{c_1}, t_{s_1} \rangle$ |
| | $\leftarrow m_{s_2}$ | | $\leftarrow t_{s_1}$ | | $\leftarrow t_{s_1}$ | |
| | $m_{c_3} \rightarrow$ | $\langle m_{c_3}, m_{s_3} \rangle$ | $t_{c_3} \rightarrow$ | $\langle t_{c_3}, t_{s_1} \rangle$ | $t_{c_1} \rightarrow$ | $\langle t_{c_1}, t_{s_1} \rangle$ |
| | $\leftarrow m_{s_3}$ | | $\leftarrow t_{s_1}$ | | $\leftarrow t_{s_1}$ | |
| **Session 3** | $m_{c_4} \rightarrow$ | $\langle m_{c_4}, m_{s_4} \rangle$ | $t_{c_1} \rightarrow$ | $\langle t_{c_1}, t_{s_1} \rangle$ | $t_{c_1} \rightarrow$ | $\langle t_{c_1}, t_{s_1} \rangle$ |
| | $\leftarrow m_{s_4}$ | | $\leftarrow t_{s_1}$ | | $\leftarrow t_{s_1}$ | |
| | $\leftarrow m_{s_5}$ | $\langle m_{s_5}, m_{c_5} \rangle$ | $\leftarrow t_{s_2}$ | $\langle t_{s_2}, t_{c_1} \rangle$ | $\leftarrow t_{s_2}$ | $\langle t_{s_2}, t_{c_2} \rangle$ |
| | $m_{c_5} \rightarrow$ | | $t_{c_1} \rightarrow$ | | $t_{c_2} \rightarrow$ | |
| | $\leftarrow m_{s_6}$ | $\langle m_{s_6}, m_{c_6} \rangle$ | $\leftarrow t_{s_3}$ | $\langle t_{s_3}, t_{c_1} \rangle$ | $\leftarrow t_{s_2}$ | $\langle t_{s_2}, t_{c_2} \rangle$ |
| | $m_{c_6} \rightarrow$ | | $t_{c_1} \rightarrow$ | | $t_{c_2} \rightarrow$ | |
| | $\leftarrow m_{s_7}$ | $\langle m_{s_6}, m_{c_6} \rangle$ | $\leftarrow t_{s_4}$ | $\langle t_{s_4}, t_{c_4} \rangle$ | $\leftarrow t_{s_2}$ | $\langle t_{s_2}, t_{c_2} \rangle$ |
| | $m_{c_7} \rightarrow$ | | $t_{c_4} \rightarrow$ | | $t_{c_2} \leftarrow$ | |

As discussed in Section II-B, alignment-based clustering methods also utilize similarity scores. However, they have to train a fixed threshold for all protocols, which cannot avoid errors due to the overlap and different score distributions of different protocols. In contrast, We use EER to describe the distribution of similarity scores and do not need a fixed threshold.

**Remote Coupling Constraints.** In the preprocessing step, we split original traces into sessions, in which we can group messages from client side and server side into pairs by their timestamps, IP, and port numbers. For example in Figure 1, we can generate message pairs as shown in Table II. After clustering by the candidate keywords of both sides, messages can be replaced with clusters they belong to and message pairs are transformed to cluster pairs. The right two columns show the cluster pairs we generate by fields $f_1$ and $f_7$, respectively. For a cluster on one side with size $N$, we count the largest number of corresponding messages on the other side that belong to a same cluster, denoted by $M$, and have the following.

$$p_r = \frac{M}{N}$$

For example, for the message type pairs of $f_1$, there are four clusters (in red) paired up with $t_{s_1}$, two of which are $t_{c_1}$. As such, the $p_r$ for cluster $t_{s_1}$ is 0.50. In Table II for $f_7$, there are only two unique cluster pairs, i.e., $\langle t_{c_1}, t_{s_1} \rangle$ and $\langle t_{s_2}, t_{c_2} \rangle$. Therefore, all clusters have their $p_r = 1$, suggesting better clustering quality than using $f_1$.

**Structure Coherence Constraints.** Structure coherence constraints state that messages of the same type share similar field structure. For messages of different types, they may share some common fields, separated by their unique fields. When aligning these messages, alignment gaps are formed due to these type-specific fields. For example in Figure 9, the two messages are of different types with different field structure. If they are wrongly put into a cluster, a lot of gaps ('-') will be inserted to make their common fields aligned. Although gaps also exist in the alignment for messages of the same type (due to data variation), the former case usually results in more gaps. Hence, after clustering with the candidate field, we align messages in the same cluster again and count the average number of alignment gaps. The proportion of gaps is used as the prior probability of coherence constraints.

$$p_s = 1 - \frac{\text{Average number of gaps in a message}}{\text{Total length of an (aligned) message}}$$

For example, there are 4 messages $m_{c_0}$, $m_{c_2}$, $m_{c_3}$, and $m_{c_4}$ in cluster $t_{c_1}$ of field $f_7$ in Figure 7b. Based on the MSA results shown in Figure 4a, messages $m_{c_0}$ and $m_{c_4}$ have 11 gaps after alignment, denoted by the symbols '-' inserted at the tail after alignment. In contrast, $m_{c_2}$ and $m_{c_3}$ have no gap. After alignment (and gap insertion), all the four messages have the length of 28. Hence the average number of gaps is $(11+0+0+11)/4 = 5.5$ for $t_{c_1}$ and $p_s$ for the cluster is computed as $1 - 5.5/28$.

**Dimension Constraints.** We consider two metrics in dimension constraints: the total number of clusters and the number of *single-message clusters*, in which there is only a single message.

The first metric is defined as follows.

$$r_{distinct\_value} = \frac{\text{Number of distinct field values}}{\text{Number of messages}}$$

We compare it with a threshold $t_{value}$, which is conservatively set to 0.5 in this paper. If the metric is greater than the threshold, it means that the candidate field generates too many clusters, which is less likely to be a true keyword. Note that a true keyword usually has only a small number of distinct values. Thus 0.5 is a very conservative value to make sure the true keyword will not be ignored and it doesn't affect the number of generated clusters.

The second metric is the proportion of single-message clusters over the total number of clusters.

$$r_{single\_cluster} = \frac{\text{Number of single-message clusters}}{\text{Number of clusters}}$$

It is also compared against a threshold $t_{single}$, which is 0.5 as well in this paper. If both values are smaller than their thresholds, the dimension constraint is given a high probability, e.g., 0.95. Otherwise it is set a low probability, e.g., 0.1.

$$p_d = \begin{cases} 0.95, & \text{if } r_{distinct\_value} < t_{value} \\ & \text{and } r_{single\_cluster} < t_{single} \\ 0.1, & \text{otherwise} \end{cases}$$

From the clustering results shown in Figure 7, we can decide that $r_{single\_cluster}$ for field $f_1$ is $5/8$, thus its $p_d$ is 0.1, whereas $f_7$ satisfies both conditions and its $p_d$ is 0.95.

**Normalization.** As discussed above, the four observation constraints are represented by different metrics, which do not mean general probabilities and may have different distributions. For example, EER is usually in range $[0.3, 0.6]$, while the computed $p_r$ for remote coupling constraints could be as high as 1. If probabilities of one type of observation constraint are limited in a small range, this type of observation constraint may play a less important role compared with others. To avoid this issue, we normalize probabilities of the same type of constraints for all candidate fields to the same range, e.g., $[0.1, 0.95]$, before further probabilistic inference.

### C. Probabilistic Inference

In this stage, all the constraints are considered together to form a joint distribution. Let boolean variable $k$ denote the keyword predicate and $x_i$ denote the observation predicates in Table I. Then all constraints can be represented as probabilistic functions with boolean variables. Specifically, an observation constraint $x_i = 1(p)$ is translated as follows.

$$f(x_i) = \begin{cases} p, & \text{if } x_i \text{ is true} \\ 1-p, & \text{otherwise} \end{cases}$$

And an inference constraint $k \xrightarrow{p_\rightarrow} x_i$ is translated as follows.

$$f(k, x_i) = \begin{cases} p_\rightarrow, & \text{if } k \rightarrow x_i \text{ is true} \\ 1-p_\rightarrow, & \text{otherwise} \end{cases}$$

Inference constraint $k \xleftarrow{p_\leftarrow} x_i$ is similarly transformed. Then the conjunction of all the constraints can be denoted as the product of all the corresponding probabilistic functions:

$$f(k, x_1, x_2, \ldots, x_n) = f_1 \times f_2 \times \cdots \times f_m$$

The joint probability function is defined as follows [53].

$$p(k, x_1, x_2, \ldots, x_n) = \frac{f_1 \times f_2 \times \cdots \times f_m}{\sum_{k, x_1, \ldots, x_n} (f_1 \times f_2 \times \cdots \times f_m)}$$

Our interest is the marginal probability of the assumption $k$, which is the sum over all observation variables. This value represents the probability that the candidate field is the keyword.

$$p(k) = \sum_{x_1, \ldots, x_n} p(k, x_1, x_2, \ldots, x_n)$$

**Factor Graph.** Due to the large number of constraints, the computation of the marginal probability is very expensive. We use a graphical model, *factor graph* [86], to represent all probabilistic functions and conduct efficient computation. A factor graph is a bipartite graph with two kinds of nodes, i.e., factor nodes and variable nodes. Factor nodes represent probabilistic functions. Variable nodes represent the variables used in probabilistic functions with edges connected to the corresponding factor nodes. Then the sum-product belief propagation algorithm [53] is used to compute the marginal probability of a node by iterative message passing in an efficient way. Intuitively, one can consider this as a rumor spreading procedure. The observations are initial rumors. In each iteration, each variable (think of it as a person) collects all the rumors about itself from its neighbors, aggregates them, and passes the aggregated rumor on to the connected factors. Each factor (involving multiple variables) collects the rumors of its variables and computes marginal probabilities based on the conditional probabilities denoted by the factor and then propagates the computed probabilities to its variables. The process repeats until convergence. We are using an off-the-shelf factor graph engine [17]. The details are hence elided.

## V. Evaluation

A few protocol reverse engineering works have been proposed to cluster messages based on network traces. However, their evaluation studies are inadequate in a number of places. Most works only conduct experiments on a small number of protocols with the focus on text protocols. As discussed earlier, it is usually more difficult to cluster binary protocols. Most works rely on sensitive parameters which need to be adjusted for different protocols. Hence, they ought to be evaluated against more protocols to illustrate effectiveness and generality. Another common issue is that most existing works do not make their systems publicly available, nor do they use public datasets. This makes it hard to validate these methods or conduct comparative studies.

As binary analysis and network trace based techniques have different application scenarios and none of binary analysis techniques is publicly available, it is difficult to compare NETPLIER with binary analysis techniques. Hence, our comparative studies focus on existing network trace based techniques. In this section, we compare NETPLIER with two state-of-the-art methods, Netzob and Discoverer, and show the advantage of NETPLIER with experiments on clustering of different protocols and datasets of different sizes, format inference, and state machine inference (Section V-A - Section V-D).

Internet of Things (IoT) devices are increasingly popular today. The evaluation of existing protocol reverse engineering works usually focus on well-known application layer protocols, while IoT devices often have customized or self-defined protocols for wireless communication. To validate the generality of NETPLIER, we also compare with AWRE [69], a recent work for the physical layer of proprietary wireless protocols (Section V-E), and conduct evaluation with multiple unknown protocols used in real IoT devices (Section V-F).

## A. Experiment Setup

**Datasets.** We construct our datasets from several publicly available traces [66], [41], [9], [5], [11], [14]. We filter messages of 10 common protocols from these traces with focus on binary protocols. Note that we cover most protocols tested by existing works, while each existing work usually only tested a small part of these protocols. For each protocol, we filter at least 1000 messages except TFTP due to the lack of enough messages. Table III shows the statistical information of the datasets. These protocols represent different categories. FTP is a common text protocol. DHCP has complex field structures which lead to low message similarities. ICMP and NTP are simple in structure but may contain broadcast messages, which leads to fewer coupling constraints. SMB and SMB2 are two versions with different field structures and both have many message types, as shown in Table III. TFTP is used for file transfer and its messages may vary a lot in length. ZeroAccess is a P2P botnet protocol, which is a representative of command and control protocols. DNP3 and Modbus are two commonly used protocols in industrial control systems. The variety of these protocols shows the generality of our method.

**Implementation.** In NETPLIER, we use MAFFT [46] for multiple sequence alignment and pgmpy [17] for probabilistic inference. As mentioned before, most existing works are not open-sourced. Hence we re-implement the two representative clustering methods discussed in Section II, Netzob and Discoverer, for comparative studies. We implement Netzob on its underlying framework [7] and implement Discoverer based on a through study of its paper. The parameters are chosen following Bossert's work [25] and trained on small datasets with 100 messages. As only partial data of Netzob are public and Discoverer used proprietary datasets, it is hard to compare with original works. However, we test our implementations on the datasets used in Netzob and achieve similar results, which provides validation of the correctness of our re-implementation.

TABLE III: Dataset information

| Protocol | # Message |  |  | # Message Types |  | # Session |
|---|---|---|---|---|---|---|
|  | Client | Server | Total | Client | Server |  |
| DHCP | 523 | 477 | 1000 | 3 | 2 | 100 |
| DNP3 | 460 | 540 | 1000 | 3 | 3 | 40 |
| FTP | 458 | 542 | 1000 | 14 | 15 | 30 |
| ICMP | 492 | 508 | 1000 | 1 | 2 | 73 |
| Modbus | 494 | 506 | 1000 | 4 | 4 | 13 |
| NTP | 678 | 322 | 1000 | 3 | 1 | 83 |
| SMB | 454 | 546 | 1000 | 9 | 10 | 89 |
| SMB2 | 510 | 490 | 1000 | 14 | 15 | 242 |
| TFTP | 225 | 228 | 453 | 4 | 1 | 34 |
| ZeroAccess | 577 | 433 | 1000 | 1 | 1 | 278 |

## B. Evaluation of Clustering

**Evaluation Metrics.** Some non-keyword fields may play the same role as a keyword and also generate correct clusters. Thus, the evaluation is focused on the clustering results instead of the keyword identification. Existing works use different metrics in their experiments to evaluate clustering results and most of them have similar meanings. In this paper, we use common objectives for clustering performance evaluation, which are called *homogeneity* and *completeness* [71]. Homogeneity means that each cluster contains only messages of a single message type, while completeness means all messages of a given type are assigned to the same cluster. We use two scores to measure homogeneity and completeness, denoted as $h$ and $c$, respectively. The two scores are computed using conditional entropy analysis. Specifically, let $n$ denote the total number of messages, $n_t$ and $n_c$ denote the number of messages belonging to message type $t$ and cluster $c$, and $n_{t,c}$ denote the number of messages from type $t$ assigned to cluster $c$. Then the entropy of the types ($H(T)$) is defined as:

$$H(T) = -\sum_{t=1}^{|T|} \frac{n_t}{n} * \log \frac{n_t}{n}$$

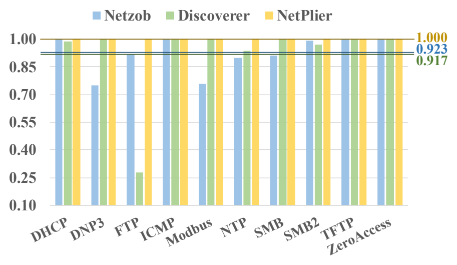
And the conditional entropy of the types given the cluster assignments is defined as:

$$H(T|C) = -\sum_{t=1}^{|T|} \sum_{c=1}^{|C|} \frac{n_{t,c}}{n} * \log \frac{n_{t,c}}{n_c}$$

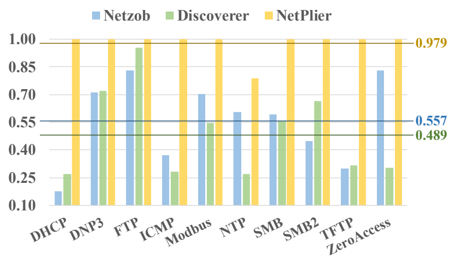
The entropy of the clusters ($H(C)$) and the conditional entropy of clusters given type ($H(C|T)$) are defined in a symmetric way. Then scores $h$ and $c$ are computed as:

$$h = 1 - \frac{H(T|C)}{H(T)}$$

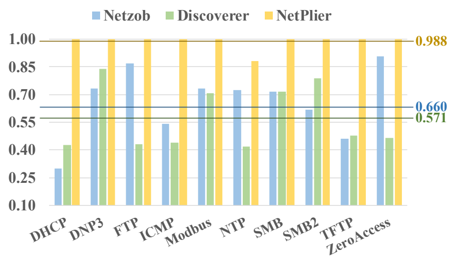
$$c = 1 - \frac{H(C|T)}{H(C)}$$



(a) Clustering results in homogeneity

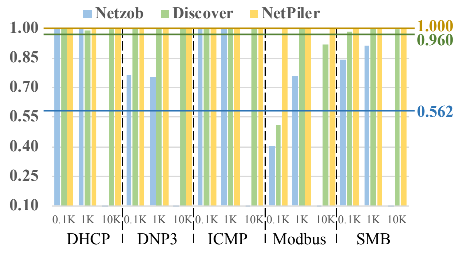


(b) Clustering results in completeness

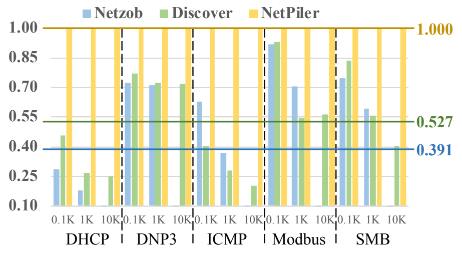


(c) Clustering results in V-measure

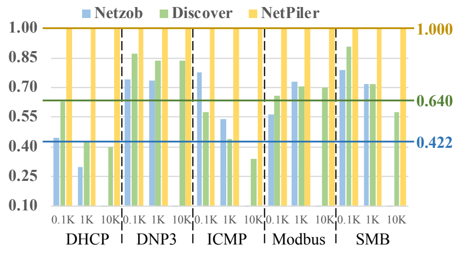
Fig. 10: Clustering result



(a) Clustering results in homogeneity



(b) Clustering results in completeness



(c) Clustering results in V-measure

Fig. 11: Clustering result on datasets of different sizes

The two scores range from 0 to 1 and the higher the better. To consider the two metrics together, we also introduce their harmonic mean, which is called *V-measure*. The score of V-measure ($v$) can be computed as:

$$v = 2 * \frac{h * c}{h + c}$$

In the following experiments, we will compute the three metrics to measure the clustering results.

**Results of Different Protocols.** We compare our method with Netzob and Discoverer on different protocols. As Netzob and Discoverer only consider messages from one side, we use them to cluster messages of the client side and server side separately, and then compute metrics with all clusters, while NETPLIER infers the keywords of both sides at the same time and its results consider all messages already.

NETPLIER identifies the keyword after two rounds of the iterative alignment and clustering for DHCP, and uses only one round for other protocols. This is due to the complex field structures of DHCP, which causes some alignment errors in the first round. The clustering results of different protocols are shown in Figure 10. NETPLIER substantially outperforms Netzob and Discoverer for all protocols. Homogeneity and completeness are determined by correctly recovering message types. Since NetPlier recognizes keywords correctly, both metrics are 100%, which is the advantage of NetPlier. The only exception is NTP, for which NETPLIER generates a few more clusters and gets a completeness score of 0.788. This is because NTP uses several bits representing its keyword, while the minimal keyword candidate generated in NETPLIER is a byte. Nonetheless, NETPLIER still outperforms Netzob and Discoverer clearly. Netzob and Discoverer have similar performance. Although they perform well in homogeneity, their completeness scores are much lower. As we discussed before, Netzob and Discoverer are not able to identify the exact number of clusters. They are sensitive to their parameters and make deterministic decisions in the presence of uncertainty, which makes it hard to balance both homogeneity and completeness. Hence they usually generate more clusters to make sure the accuracy, which leads to a low completeness score.

**Datasets of Different Sizes.** Besides different protocol types, the protocol reverse engineering methods may also be affected by the data sizes. To show the stability of NETPLIER, we also compare the results of datasets with different sizes. We choose five common protocols with enough messages and construct three datasets with different sizes (100, 1000, and 10000 messages) for each protocol. Figure 11 shows the clustering results on these datasets. We can see that NETPLIER performs stably on different sizes with most scores being 1. For DHCP of 10000 messages, NETPLIER's performance on completeness drops slightly (0.993) due to the complex option fields. Note that Netzob could not handle the datasets of 10000 messages due to the exponential complexity and huge memory consumption of its pair-wise alignment. In general, when the number of messages increases, the homogeneity of Netzob and Discoverer stays in the same level or increases slightly, while the completeness decreases obviously. This shows that Netzob and Discoverer are not stable for inputs of different sizes even for the same protocol.

All experiments were conducted on a server equipped with 32-cores CPU (Intel® Xeon$^{\text{TM}}$ E5-2690 @ 2.90GHz) and 128G main memory. Table IV shows the execution time and maximum memory on datasets of 1000 messages. NETPLIER and Discoverer also generate formats of each cluster at the same time, while Netzob only conducts clustering. NETPLIER consumes similar memory resource to Discoverer and is much less than Netzob. Note that Netzob consumes lots of memory and it stops execution for datasets with 10000 messages as shown in Figure 11. The bottleneck of NETPLIER lies in MSA, as we use iterative refinement in MSA and constraints generation. The time complexity of MSA could vary a lot for different protocols. For well-formatted protocols, e.g., DNP3,

TABLE IV: Overhead measurement. The unit of time is *min* and the unit of memory is *MB*.

| Protocol | Netzob | | Discoverer | | NetPlier | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| | | | | | MSA | | Constraints Generation | | Probabilistic Inference | |
| | Time | Memory | Time | Memory | Time | Memory | Time | Memory | Time | Memory |
| DHCP | 17.092 | 124.420 | 0.034 | 13.973 | 82.555 | 0.059 | 4.645 | 14.882 | 9.996 | 5.411 |
| DNP3 | 1.361 | 104.282 | 0.002 | 0.519 | 4.598 | 0.059 | 1.207 | 15.208 | 25.891 | 60.965 |
| FTP | 1.549 | 103.815 | 0.002 | 0.210 | 20.668 | 0.059 | 5.862 | 15.365 | 103.022 | 58.962 |
| ICMP | 1.735 | 102.543 | 0.005 | 0.995 | 6.780 | 0.059 | 0.385 | 14.829 | 2.055 | 8.919 |
| Modbus | 1.357 | 99.336 | 0.004 | 1.268 | 7.384 | 0.059 | 0.660 | 14.894 | 4.781 | 20.439 |
| NTP | 2.090 | 122.784 | 0.013 | 2.510 | 6.171 | 0.059 | 6.402 | 17.681 | 242.330 | 106.033 |
| SMB | 1.917 | 109.549 | 0.015 | 3.687 | 23.053 | 0.059 | 6.348 | 15.481 | 122.812 | 61.950 |
| SMB2 | 5.803 | 114.231 | 0.017 | 4.176 | 39.392 | 0.059 | 9.628 | 15.048 | 37.799 | 31.890 |
| TFTP | 3.299 | 34.400 | 0.049 | 30.966 | 83.585 | 0.059 | 0.466 | 4.120 | 0.044 | 1.336 |
| ZeroAccess | 19.258 | 109.291 | 0.072 | 32.571 | 59.127 | 0.059 | 2.332 | 18.163 | 0.396 | 1.170 |

it is close to $O(N * L)$, where $N$ is the number of messages and $L$ is the length of a message. However, for complex protocols, e.g., with many variable-length fields, the worst case is $O(L * N^2)$. The time complexity for constraints generation is $O(N^2)$ as we need to compare each two messages for similarity. The time for probabilistic inference is determined by the number of fields which does not grow with the dataset sizes. Although NetPlier executes slower than the other two baselines due to the need of aligning complex messages and probabilistic inference in datasets of 1000 messages, we argue that the overhead is reasonable as it is an offline technique and hence one-time effort. Also, as discussed above, NetPlier is not sensitive to data sizes, which means the overhead could be improved by executed on smaller datasets.

## C. Evaluation of Format Inference

To show the benefits of our clustering results, we further infer the field structures. The clustering results of Discover and NetPlier already contain the format information. Netzob's format inference is based on its pairwise alignment [25]. However, it has to consider the alignment results of all pairs in a cluster at the same time. As such, its format inference can handle fewer messages than the clustering stage. We utilize tshark [12] to obtain the ground truth, i.e., the information of true fields. Then for each inferred field, we compare its boundaries and values with true fields. We consider an inferred field as a *correct* one if the inferred field is part of a single true field or combines several consecutive true fields. Specifically, the field is *accurate* if it perfectly matches a true field. However, an inferred field is considered to be *incorrect* if it contains multiple incomplete true fields. It is also incorrect if a dynamic field is mistaken as static. For example, as discussed above, prior works usually generate more clusters to improve the homogeneity, so messages of the same type may be placed into multiple clusters. Some fields may be considered as static as all messages in the cluster have the same value. However, messages of the same type in other clusters may have different values, which means they are actually dynamic fields. Note that $h$, $c$, and $v$ scores are common metrics used in clustering when having ground truth labels. They cannot be directly applied to measuring the results of format and state machine inferences. Then two metrics, correctness and perfection, are computed to measure the inferred formats, which are defined as follows:

$$correctness = \frac{\text{Number of correct fields}}{\text{Number of total inferred fields}}$$

TABLE V: Evaluation of format inference

| Protocol | Netzob | | Discoverer | | NetPlier | |
|---|---|---|---|---|---|---|
| | Corr. | Perf. | Corr. | Perf. | Corr. | Perf. |
| DHCP | 0.089 | 0.000 | 0.768 | 0.016 | 0.994 | 0.014 |
| DNP3 | 0.702 | 0.099 | 0.486 | 0.018 | 0.752 | 0.183 |
| FTP | 1.000 | 1.000 | 1.000 | 1.000 | 1.000 | 1.000 |
| ICMP | 0.571 | 0.144 | 0.259 | 0.102 | 0.972 | 0.090 |
| Modbus | 0.587 | 0.084 | 0.344 | 0.049 | 0.698 | 0.049 |
| NTP | 0.830 | 0.000 | 0.661 | 0.000 | 0.851 | 0.000 |
| SMB | 0.660 | 0.152 | 0.608 | 0.207 | 0.964 | 0.237 |
| SMB2 | 0.349 | 0.003 | 0.793 | 0.041 | 0.923 | 0.069 |
| TFTP | 0.666 | 0.454 | 0.147 | 0.000 | 0.986 | 0.009 |
| ZeroAccess | N/A | N/A | 0.155 | 0.000 | 0.980 | 0.000 |

$$perfection = \frac{\text{Number of accurate fields}}{\text{Number of total true fields}}$$

Table V shows the results of format inference. As textual protocols could utilize delimiters to generate fields, all methods can achieve 100% correctness and perfection on FTP. For binary protocols, we can see that our clustering results obviously improve the correctness of format inference, which means it is more likely to generate valid messages based on our inferred formats. Due to the nature of network trace based techniques, the perfection of all techniques tends to be low. For example, all the ZeroAccess messages in our datasets have the same value for some bytes in a true field. These bytes will be separated as static fields, which is correct but not perfect. So both Discover and our method achieve 0% perfection. The largest ZeroAccess cluster generated by Netzob contains more than 500 messages, which is beyond its handling capacity. Thus Netzob fails to generate formats for ZeroAccess due to timeout. Note that correct field formats can still generate valid messages even if they are not accurate.

## D. Evaluation of State Machine Inference

After clustering, we can further infer the finite state machine (FSM). The effectiveness of FSM inference is determined by the clustering results. Low completeness of clustering leads to excessive states, making state machine too complex to provide useful information. Table VI shows the number of inferred FSM states. The ground truth is computed from the specification. Compared with Netzob and Discoverer,

TABLE VI: Number of states

| Protocol | Groud truth | Netzob | Discoverer | NETPLIER |
|---|---|---|---|---|
| DHCP | 8 | 77 | 56 | 8 |
| DNP3 | 11 | 11 | 11 | 11 |
| FTP | 11 | 20 | 11 | 11 |
| ICMP | 10 | 12 | 20 | 10 |
| Modbus | 42 | 42 | 90 | 42 |
| NTP | 69 | 152 | 179 | 137 |
| SMB | 16 | 89 | 58 | 16 |
| SMB2 | 111 | 327 | 126 | 111 |
| TFTP | 4 | 4 | 4 | 4 |
| ZeroAccess | 6 | 12 | 97 | 6 |

NETPLIER always generates fewer states, and for most protocols, it has the same number of states as the ground truth, including those that are very complex. In contrast, Netzob and Discoverer could generate 2 to 3 times more states. Note the Discoverer identifies more keywords than those indicating message types. As such, on one hand, it generates smaller clusters that may denote message subtypes. On the other hand, the overly fine-grained information creates troubles for downstream applications such as format and state-machine inference as suggested by our results. This indicates correctly recognizing message types is critical. NETPLIER is designed to serve that purpose.

## *E. Evaluation of Other Layer Protocols*

As discussed earlier, existing protocol reverse engineering works usually focus on application layer protocols. However, in wireless communication, physical layer protocols could also be proprietorially designed, where existing works cannot be applied to as physical layer protocols are binary. AWRE [69] is designed for field inference of physical layer protocols. It uses prior semantic knowledge as heuristics to identify common fields in physical layer, including the Preamble Field, Sync Field, Length Field, Address Field, Sequence Number Field, and Checksum Field. We compare our method with AWRE on the eight physical layer protocols used in the paper. These protocols vary a lot in their field structures and we list their features in the second column of Table VII. The number of messages for each protocol is set to 50, which can ensure the accuracy of AWRE according to the paper. Compared with above evaluation on well-known benchmarks, fewer messages are used here as it is usually not easy to collect a large dataset of IoT devices in reality.

Our method can be applied to these physical layer protocols directly. The only difference is that the information from the network layer (i.e., timestamps, IP addresses, and port numbers as mentioned in Section III-A) is not available for physical layer protocols. We simply consider consecutive messages with different directions as a pair for remote coupling constraints. Other constraints are the same as those on the application layer protocols.

Table VII shows the clustering and format inference results of AWRE and NETPLIER. Both methods generate perfect clustering with 100% homogeneity and completeness on all eight protocols. For format inference, AWRE achieves 100% correctness and perfection for all protocols. NETPLIER also performs well for correctness and only generates a few errors

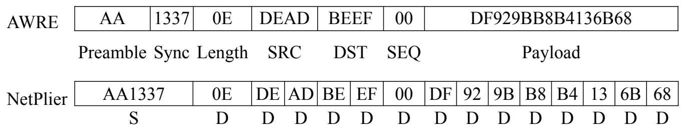


Fig. 12: Example of format inference by AWRE and NETPLIER

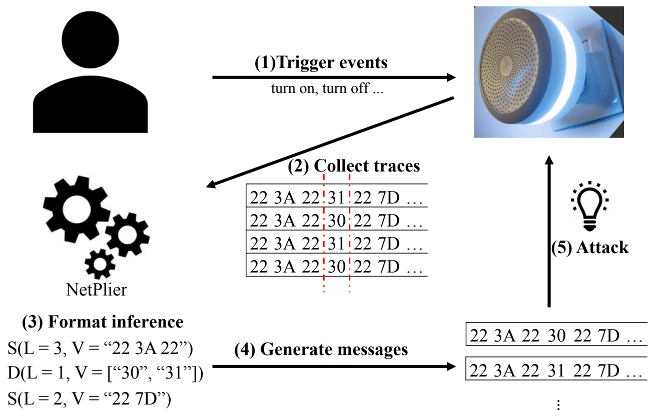


Fig. 13: Example of reverse engineering unknown protocols of IoT devices

for Protocol 7. This is because a dynamic field has the same prefix for all messages in the small dataset, and hence is considered a static field and merged with the adjacent static field. The merged one is considered incorrect following the definition in Section V-C. These errors do not affect the generation of valid messages and may be reduced if more messages are used. Also, NETPLIER achieves lower perfection than AWRE. For example in Figure 12, the inferred format of AWRE perfectly matches the true format, because AWRE assumes that the types of all fields are already known and their semantics could be used in the inference. However, NETPLIER is a general tool for all protocols and do not have such prior knowledge. For example, the Preamble and Sync fields usually have the same values shared by all messages. Thus NETPLIER considers them together as a single static field. Also, the values of dynamic fields, e.g., the Payload field, in the test protocols are randomly generated. They are hence recognized as multiple smaller fields by our method and make the perfection low. Similarly, these inferred fields are still correct and useful in practice. For example, our formats can still generate valid Preamble and Sync fields. We also validate the generated messages in Section V-F. In addition, we optimize our format inference using the same assumptions by AWRE, i.e., we model the semantics of those pre-defined field types as additional constraints. After that, our method could also achieve 100% correctness and perfection as shown in the last two columns of Table VII. Note that AWRE is only designed for physical layer protocols and could not be applied to general application protocols as those evaluated in Section V-B and Section V-C.

## *F. Evaluation of Unknown Protocols*

An important application of protocol reverse engineering is to study the customized/unknown protocols used by IoT devices. As far as we know, existing works only focus on well-known protocols as discussed in Section V-B and did

TABLE VII: Comparison with AWRE

| Protocol | | | | AWRE | | | NetPlier | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| # | Comment | # Msg. | # Msg. Types | Clustering | Format Inference | | Clustering | Format Inference | | | |
| | | | | | | | | w/o Assumption | | w/ Assumption | |
| | | | | h/c/v | Corr. | Perf. | h/c/v | Corr. | Perf. | Corr. | Perf. |
| 1 | Common protocol | 50 | 1 | 1/1/1 | 1.000 | 1.000 | 1/1/1 | 1.000 | 0.286 | 1.000 | 1.000 |
| 2 | Unusual field sizes | 50 | 1 | 1/1/1 | 1.000 | 1.000 | 1/1/1 | 1.000 | 0.143 | 1.000 | 1.000 |
| 3 | Ack, CRC8 CCITT | 50 | 2 | 1/1/1 | 1.000 | 1.000 | 1/1/1 | 1.000 | 0.385 | 1.000 | 1.000 |
| 4 | Ack, CRC16 CCITT | 50 | 3 | 1/1/1 | 1.000 | 1.000 | 1/1/1 | 1.000 | 0.167 | 1.000 | 1.000 |
| 5 | 3 participants with ack frame | 50 | 2 | 1/1/1 | 1.000 | 1.000 | 1/1/1 | 1.000 | 0.273 | 1.000 | 1.000 |
| 6 | Short address | 50 | 1 | 1/1/1 | 1.000 | 1.000 | 1/1/1 | 1.000 | 0.800 | 1.000 | 1.000 |
| 7 | 4 participants varying preamble size & sync | 50 | 3 | 1/1/1 | 1.000 | 1.000 | 1/1/1 | 0.980 | 0.190 | 1.000 | 1.000 |
| 8 | Nibble fields, LE | 50 | 2 | 1/1/1 | 1.000 | 1.000 | 1/1/1 | 1.000 | 0.250 | 1.000 | 1.000 |

TABLE VIII: Evaluation on unknown IoT protocols

| Device | Event | Message Format (Request & Response) | # Triggered Events |
|---|---|---|---|
| Nest Thermostat | Temperature Up/Down | S(32) D(27) S(2) D(36) S(39) D(30) S(9) D(3) S(40) D(4) S(11)<br>S(86) D(62) S(24) | 4/4 |
| | Fan On/Off | On: S(32) D(20) S(2) D(36) S(38) D(23) S(16) D(3) S(7)<br>Off: S(32) D(20) S(2) D(36) S(38) D(23) S(20)<br>S(77) D(62) S(29) | |
| Nest Protect | Emergency Shutoff On/Off | S(77) D(4, 6) S(91) D(0, 5) S(25) D(4, 5) S(4)<br>S(16) D(113) S(15) | 2/2 |
| Aqara Hub | On/Off | S(21) D(1) S(12) D(1) S(85)<br>S(26) D(1) S(25) D(3) S(1) D(3) S(1) D(17) S(53) | 2/2 |
| Aqara Smart Plug | On/Off | S(19) D(1) S(12) D(1) S(85)<br>S(24) D(1) S(25) D(3) S(1) D(3) S(1) D(17) S(53) | 2/2 |
| Aqara Contact Sensor | Open/Closed | S(9) D(1) S(40) D(6) S(18)<br>S(50) D(13) S(10) D(1) S(351) | 2/2 |
| Aqara Motion Sensor | Detected/ Not detected | S(9) D(1) S(40) D(6) S(18)<br>S(52) D(13) S(10) D(1) S(53) D(13) S(10) D(1) S(52) D(13) S(10) D(1) S(331) | 2/2 |

not evaluate on unknown protocols. In this section, we apply NetPlier to real IoT devices to evaluate its effectiveness.

There are several works studying the security issues of IoT devices via public traces [70], [78]. However, as the ground truth for unknown protocols is often absent, it is difficult to use public datasets and evaluate the clustering results like what we do in Section V-B. Instead, we have to conduct active evaluation by communicating with real-time devices. We set up a testbed with 6 popular IoT devices of different functionalities, including a hub (with a light), three controllers (a thermostat, a Nest Protect smoke detector, and a smart plug), and two sensors (a contact sensor and a motion sensor).

Figure 13 shows the workflow of our evaluation on unknown protocols. First, we collect the traces by manually triggering various events of the devices, which are shown in the second column of Table VIII. For the two sensors, we take the corresponding actions, e.g., opening the door, to change their states; for the other devices, we control them using their official applications on the Android smartphone. Each event is repeated for 50 times and traces are collected with a label of the event. After having the traces, we apply NetPlier to infer the message formats of each event type as discussed in Section III-E. The results are shown in the third column of Table VIII. For each event type, we consider the formats of both request and response messages. Specifically, Nest Thermostat has two request messages for turning fan on and off, respectively. Here, we only show the type ('S' for static fields and 'D' for dynamic fields) and length of each field, denoted as $Type(Length)$. Then we use the inferred formats to generate messages. For static fields, their values are already fixed. The challenge is to decide the value of dynamic fields. We consider both existing values (in the traces) and random values. For example in Figure 13, we turn on and off the light and collect four messages. After format inference, we find three fields in a cluster of request messages, including two static fields and a dynamic one. The dynamic field has only two existing values, i.e., "30" and "31", which is highly likely to indicate the on/off status and could be used to generate messages directly. In real traces, dynamic fields may have many different values, e.g., the Sequence ID. We generate random values for these fields. Finally, we validate the results by checking if the generated (request) messages could trigger the same events successfully, i.e., if we can turn on or off the light by generated messages in this example. As shown in the last column of Table VIII, all events could be triggered successfully, which validates the formats inferred by NetPlier. In Section VI-A we show a detailed case study on Nest Thermostat.

## VI. Application

In this section, we demonstrate two applications of NetPlier: Internet of Things (IoT) protocol reverse engineering and malware behavior analysis.

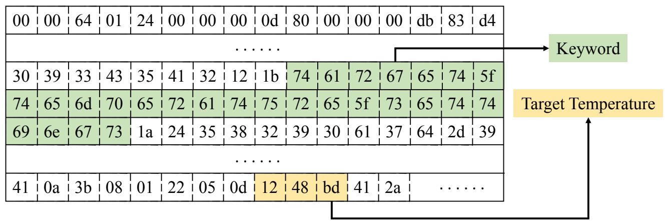
### A. *IoT Protocol Reverse Engineering*

IoT protocol analysis becomes increasingly important for IoT security. However, it is challenging to analyze IoT protocols due to the lack of specification and the limited access to source code. In this case study, we use NetPlier to analyze the protocol used by Google Nest Thermostat E [4], a commercial smart thermostat. In particular, we fake an SSL Certificate Authority (CA) and dump all Google Nest's traces. After decryption, NetPlier is used to analyze the protocol format. With the reverse engineered protocol, we successfully hijacked Google Nest to perform malicious behaviors (e.g., setting a specific indoor temperature) via sending crafted messages, which indicates the correctness of the recovered protocol format. Note that we used a fake CA to decrypt TLS data to focus our study on protocol reverse engineering. Acquiring plain-text messages with other means is beyond the scope of this paper.

Figure 14 presents a temperature-setting message in hex format. The original message has 351 bytes. Here we only present part of it and highlight the interesting fields. The keyword lies in the green field and has a variable length, which is the first dynamic field (D(27)) in the formats of Nest Thermostat shown in Table VIII. After alignment and inference, NetPlier precisely identifies the keyword field. NetPlier's correct clustering results further help us observe a one-to-one relation between the temperature and the yellow field (D(4) in red in Table VIII), allowing us to determine the semantics of the yellow field, i.e., the temperature that a user wants to set. In our experiment, manipulating this field allowed us to directly change the indoor temperature. We also successfully created messages to instruct Google Nest to turn on/off the fan and perform other human-observable behaviors.

### B. *Malware Analysis*

The proliferation of new strains of malware every year poses a prominent security threat and renders the importance of malware analysis. A popular approach to understanding malware is to run it in a sandbox. However, handling command and control (C&C) behavior is a well-known challenge, as this kind of behavior is triggered by remote servers' commands and beyond analysts' control [76]. On the other hand, most malware is equipped with C&C capabilities [42]. Hence, researchers tried to utilize protocol reverse engineering to analyze malware network trace, hoping to interpret malicious behavior [72]. We conducted a case study on leveraging NetPlier to enhance the analysis of a typical C&C botnet client (MD5: *03cfe768a8b4ffbe0bb0fdef986389dc*) which was recently reported to VirusTotal [13]. Note that the malware is packed and obfuscated, so it is difficult to analyze its behavior via static approaches. We used NetPlier to analyze its network traces (acquired by Tencent Habo [10]) and recover its state machine. Based on the recovered state machine, we simulated a client to communicate with the remote server. The procedure was iterative as the more communication is triggered



```
00 00 64 01 24 00 00 00 0d 80 00 00 00 db 83 d4
                     ......
30 39 33 43 35 41 32 12 1b 74 61 72 67 65 74 5f
74 65 6d 70 65 72 61 74 75 72 65 5f 73 65 74 74
69 6e 67 73 1a 24 35 38 32 39 30 61 37 64 2d 39
                     ......
41 0a 3b 08 01 22 05 0d 12 48 bd 41 2a    ......
```

Fig. 14: The snippet of a Google Nest's temperature-setting message

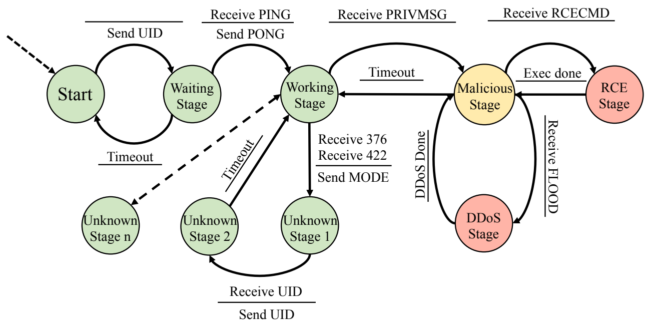


Fig. 15: Recovered state machine of the botnet client

between the client and the server, the more of the protocol can be discovered by NetPlier, which in turns allows us to trigger more.

Figure 15 demonstrates the finite state machine NetPlier recovered. Each circle denotes a state, and each direct edge denotes the transition between two states. Transition is labeled with $\frac{i}{j}$ where $i$ is the precondition of transition and $j$ is the message sent by the client. Note that for ease of understanding, we manually annotated the states after analyzing the collected syscalls in each state. The green states are those not causing direct damages, the red states are the ones containing dangerous syscalls, and the yellow ones belong to the transition period. As shown in Figure 15, when the botnet malware starts, it sends its unique id to the remote server and transits to the waiting stage. After the server verifies the id, a ping-and-pong handshake is set up to check the connection, and then the client transits to the operation stage. After that, various functionalities can be performed based on the instructions from the server. Some of the instructions are not damaging and used to setup the environment. Their details are elided. A special kind of messages with keyword *PRIVMSG* can trigger the botnet to move to the malicious stage. A few malicious behaviors like remote code execution and internal network *DDoS* are observed after the client is at this state.

## VII. Discussion

**Limitations.** Datasets of low quality are a common challenge for network trace based techniques. Information that is not included in a small dataset could not be discovered, e.g., unused message types. However, we argue that NetPlier can make better use of traces by considering multiple constraints. In Section V-B, all the datasets are collected from real-world systems, which are considered more challenging. Also, we show that NetPlier is stable even on datasets of small sizes.

Network trace based protocol reverse engineering methods are limited to unencrypted traces. A possible solution is to use a man in the middle proxy with trusted credentials, e.g., Fiddler [3] and Burp Suite [1]. It could also be combined with program analysis based protocol reverse engineering methods.

Another limitation of NETPLIER is the growing complexity and potential errors of multiple sequence alignment algorithms for larger data. Some heuristic solutions have been proposed to improve the execution speed, e.g., the combination of progressive alignment and iterative refinement [20], [77]. Also, as discussed in Section V-B, NETPLIER performs stably on different sizes and achieves similar results, which means that the speed and accuracy could be improved by using NETPLIER in several small datasets instead of the whole large one. We leave this improvement to future work.

**Generality.** Most network trace-based techniques are designed for textual protocols at the application layer. In Section V, we show that our method works well for binary protocols, physical layer protocols where network layer information is missing, and unknown protocols used in real IoT devices.

We address the problem of clustering by identifying keywords in bytes. Some protocols may use sub-byte fields as the keyword, e.g., NTP. Our results in Section V-B shows that the homogeneity of NetPlier is not affected by such fields and the completeness degrades a little. It still outperforms others. Such keyword fields could be better handled by detecting if sub-byte fields are used in the preprocessing stage. If so, the granularity of keyword candidates could be set to bits instead of bytes. We leave it to future work.

Some protocols may include uni-direction messages, e.g., broadcast messages without response, where the remote coupling constraints would be ineffective. In our experience, without using two-way messages, we will encounter some degradation in the results. However, NetPiler still outperforms the baselines due to its way of aggregating other constraints.

**Future Work.** We focus on clustering and only use a simple strategy for format inference. Heuristics for semantic information could be introduced to improve the results of format inference [26], [54], [69]. We could also apply probabilistic inference in this stage, for example, to infer potential field boundaries of consecutive variable-length fields with probabilistic constraints (e.g., that expose fields dictating runtime length values). We leave it to future work.

## VIII. RELATED WORK

**Protocol Reverse Engineering.** Protocol reverse engineering targets at inferring the specification of unknown network protocols for further security evaluation [56], [63], [37], [73]. There are two main categories, either by program analysis [28], [57], [82], [33], [59], [32] or by network traces [22], [55], [35], [52], [81], [51], [80], [26], [38], [47]. Network trace methods are usually based on sequence alignment algorithms [64] or token patterns, and are limited for their low accuracy or conciseness. In this paper, we conduct comparative studies to show the obvious improvement of NETPLIER. Token-based methods [35], [80] search for representative tokens by statistics and use them for clustering. It was shown that Discoverer outperforms these techniques as they tend to generate redundant tokens and hence clusters. IoT protocol fingerprinting technique such as PINGPONG [78] is different from protocol reverse engineering. The former leverages meta data while the latter aims to recognize message types, formats, and state machine. In fact, PINGPONG collects fingerprints on encrypted messages.

**Probabilistic Inference in Security Applications.** In recent years, probabilistic techniques [16], [85] have been increasingly used in security applications. Lin *et al.* introduce probabilistic inference into reverse engineering [58]. Different from us, they focus on memory forensic. Dietz *et al.* also leverage probabilistic inference to localize source code bugs [36]. Besides, probabilistic techniques are widely used for binary analysis [87], [61], physical unit security [45], program enhancement [49], and vulnerability detection [36], [58]. To the best of our knowledge, NETPLIER is the first approach that enforces probabilistic analysis on protocol reverse engineering. Unlike previous methods using deterministic techniques, NETPLIER gathers all possible hints from protocol behaviors and uses a systematic way of integrating them in the presence of uncertainty.

**Malware Analysis.** The proliferation of malware in the past years raises researchers' attention on detecting, analyzing, and preventing malware. Mainstream malware analysis techniques, including VirusTotal [13], Cuckoo [2], Habo [10], Padawan [8], and X-Force [67], [85], leverage the sandbox-based execution technique to obtain malicious behaviors. However, traditional behavioral-based approaches are limited on low-level syscall tracing and can rarely understand high-level semantics behaviors (e.g., performing as a backdoor). NETPLIER, on the other hand, works on collected network trace and is able to recover informative state machines, benefiting future analysis. We believe NETPLIER is complementary to these existing works.

## IX. CONCLUSION

We propose a novel probabilistic network trace based protocol reverse engineering technique. It models the inherent uncertainty of the problem by introducing random variables to denote the likelihood of individual fields representing the message type. A joint distribution can be formed between these random variables and observations made from the message samples. Probabilistic inference is used to compute the marginal posterior probabilities, allowing us to identify the message type. Messages are then precisely clustered by their types, leading to high quality reverse engineering results. Our experiments show that our technique substantially outperforms two state-of-the-art techniques Netzob and Discoverer and facilitates IoT protocol analysis and malware analysis.

## ACKNOWLEDGMENT

We thank the anonymous reviewers for their valuable comments and suggestions. We also thank Guannan Wei for his help in illustration. This research was supported in part by NSF 1901242 and 1910300, ONR N000141712045, N000141410468 and N000141712947, and IARPA TrojAI W911NF-19-S-0012. Any opinions, findings, and conclusions in this paper are those of the authors only and do not necessarily reflect the views of our sponsors.

## References

[1] "Burp suite," https://portswigger.net/burp.

[2] "Cuckoo," https://cuckoosandbox.org/.

[3] "Fiddler," https://www.telerik.com/fiddler.

[4] "Google," https://store.google.com/product/nest_thermostat_e.

[5] "Modbus trace," https://github.com/ITI/ICS-Security-Tools/blob/master/pcaps/bro/modbus/modbus.pcap.

[6] "Netplier," https://github.com/netplier-tool/NetPlier.

[7] "Netzob," https://github.com/netzob/netzob.

[8] "Padawan," https://padawan.s3.eurecom.fr/about.

[9] "Smia2011," ftp://download.iwlab.foi.se/dataset/smia2011/.

[10] "Tencent habo," https://habo.qq.com/.

[11] "Tftp trace," https://asecuritysite.com/forensics/pcap?infile=tftp.pcap.

[12] "Tshark," https://www.wireshark.org/docs/man-pages/tshark.html.

[13] "Virustotal," https://www.virustotal.com/gui/home/upload.

[14] "Zeroaccess trace," http://contagiodump.blogspot.com/ .

[15] A. Abdou, D. Barrera, and P. C. Van Oorschot, "What lies beneath? analyzing automated ssh bruteforce attacks," in *International Conference on PASSWORDS*, 2015, pp. 72–91.

[16] A. Aguirre, G. Barthe, L. Birkedal, A. Bizjak, M. Gaboardi, and D. Garg, "Relational reasoning for markov chains in a probabilistic guarded lambda calculus," in *European Symposium on Programming*, 2018, pp. 214–241.

[17] A. Ankan and A. Panda, "pgmpy: Probabilistic graphical models using python," in *Proceedings of the 14th Python in Science Conference (SCIPY)*, 2015.

[18] M. Antonakakis, T. April, M. Bailey, M. Bernhard, E. Bursztein, J. Cochran, Z. Durumeric, J. A. Halderman, L. Invernizzi, M. Kallitsis *et al.*, "Understanding the mirai botnet," in *26th USENIX Security Symposium (USENIX Security)*, 2017, pp. 1093–1110.

[19] T. Bao, R. Wang, Y. Shoshitaishvili, and D. Brumley, "Your exploit is mine: Automatic shellcode transplant for remote exploits," in *2017 IEEE Symposium on Security and Privacy (SP)*, 2017, pp. 824–839.

[20] G. J. Barton and M. J. Sternberg, "A strategy for the rapid multiple alignment of protein sequences: confidence levels from tertiary structure comparisons," *Journal of Molecular Biology*, vol. 198, no. 2, pp. 327–337, 1987.

[21] N. E. Beckman and A. V. Nori, "Probabilistic, modular and scalable inference of typestate specifications," in *Proceedings of the 32nd ACM SIGPLAN Conference on Programming Language Design and Implementation*, 2011, pp. 211–221.

[22] M. A. Beddoe, "Network protocol analysis using bioinformatics algorithms," *Toorcon*, 2004.

[23] L. Bilge, D. Balzarotti, W. Robertson, E. Kirda, and C. Kruegel, "Disclosure: detecting botnet command and control servers through large-scale netflow analysis," in *Proceedings of the 28th Annual Computer Security Applications Conference*, 2012, pp. 129–138.

[24] K. Borgolte, C. Kruegel, and G. Vigna, "Delta: automatic identification of unknown web-based infection campaigns," in *Proceedings of the 2013 ACM SIGSAC Conference on Computer & Communications Security*, 2013, pp. 109–120.

[25] G. Bossert, "Exploiting semantic for the automatic reverse engineering of communication protocols," Ph.D. dissertation, 2014.

[26] 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*, 2014, pp. 51–62.

[27] G. E. Box and G. C. Tiao, *Bayesian inference in statistical analysis*. John Wiley & Sons, 2011, vol. 40.

[28] J. Caballero, H. Yin, Z. Liang, and D. Song, "Polyglot: Automatic extraction of protocol message format using dynamic binary analysis," in *Proceedings of the 14th ACM Conference on Computer and Communications Security*, 2007, pp. 317–329.

[29] Y. Cao, Y. Shoshitaishvili, K. Borgolte, C. Kruegel, G. Vigna, and Y. Chen, "Protecting web-based single sign-on protocols against relying party impersonation attacks through a dedicated bi-directional authenticated secure channel," in *International Workshop on Recent Advances in Intrusion Detection*, 2014, pp. 276–298.

[30] R. Cappelli, D. Maio, D. Maltoni, J. L. Wayman, and A. K. Jain, "Performance evaluation of fingerprint verification systems," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 28, no. 1, pp. 3–18, 2005.

[31] Y. Chen, D. Mu, J. Xu, Z. Sun, W. Shen, X. Xing, L. Lu, and B. Mao, "Ptrix: Efficient hardware-assisted fuzzing for cots binary," in *Proceedings of the 2019 ACM Asia Conference on Computer and Communications Security*, 2019, pp. 633–645.

[32] C. Y. Cho, D. Babic, P. Poosankam, K. Z. Chen, E. X. Wu, and D. Song, "Mace: Model-inference-assisted concolic exploration for protocol and vulnerability discovery," in *20th USENIX Security Symposium (USENIX Security)*, vol. 139, 2011.

[33] P. M. Comparetti, G. Wondracek, C. Kruegel, and E. Kirda, "Prospex: Protocol specification extraction," in *30th IEEE Symposium on Security and Privacy (SP)*, 2009, pp. 110–125.

[34] A. Cozzie, F. Stratton, H. Xue, and S. T. King, "Digging for data structures," in *8th USENIX Symposium on Operating Systems Design and Implementation*, vol. 8, 2008, pp. 255–266.

[35] W. Cui, J. Kannan, and H. J. Wang, "Discoverer: Automatic protocol reverse engineering from network traces," in *16th USENIX Security Symposium (USENIX Security)*, 2007, pp. 1–14.

[36] L. Dietz, V. Dallmeier, A. Zeller, and T. Scheffer, "Localizing bugs in program executions with graphical models," in *Advances in Neural Information Processing Systems*, 2009, pp. 468–476.

[37] J. Duchene, C. Le 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, pp. 53–68, 2018.

[38] O. Esoul and N. Walkinshaw, "Using segment-based alignment to extract packet structures from network traces," in *2017 IEEE International Conference on Software Quality, Reliability and Security (QRS)*, 2017, pp. 398–409.

[39] D.-F. Feng and R. F. Doolittle, "Progressive sequence alignment as a prerequisitetto correct phylogenetic trees," *Journal of Molecular Evolution*, vol. 25, no. 4, pp. 351–360, 1987.

[40] E. Hoque, O. Chowdhury, S. Y. Chau, C. Nita-Rotaru, and N. Li, "Analyzing operational behavior of stateful protocol implementations for detecting semantic bugs," in *2017 47th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*, 2017, pp. 627–638.

[41] O. Igbe, I. Darwish, and T. Saadawi, "Deterministic dendritic cell algorithm application to smart grid cyber-attack detection," in *2017 IEEE 4th International Conference on Cyber Security and Cloud Computing (CSCloud)*, 2017, pp. 199–204.

[42] A. Islam and Z. Bu, "Classifying sets of malicious indicators for detecting command and control communications associated with malware," Apr. 25 2017, US Patent 9,635,039.

[43] V. Jain, S. Rawat, C. Giuffrida, and H. Bos, "Tiff: using input type inference to improve fuzzing," in *Proceedings of the 34th Annual Computer Security Applications Conference*, 2018, pp. 505–517.

[44] S. Jero, M. L. Pacheco, D. Goldwasser, and C. Nita-Rotaru, "Leveraging textual specifications for grammar-based fuzzing of network protocols," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 33, 2019, pp. 9478–9483.

[45] S. Kate, J.-P. Ore, X. Zhang, S. Elbaum, and Z. Xu, "Phys: probabilistic physical unit assignment and inconsistency detection," in *Proceedings of the 2018 26th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, 2018, pp. 563–573.

[46] K. Katoh, K. Misawa, K.-i. Kuma, and T. Miyata, "Mafft: a novel method for rapid multiple sequence alignment based on fast fourier transform," *Nucleic Acids Research*, vol. 30, no. 14, pp. 3059–3066, 2002.

[47] 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 (WOOT)*, 2018.

[48] P. Kohli and P. H. Torr, “Dynamic graph cuts for efficient inference in markov random fields,” *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 29, no. 12, pp. 2079–2088, 2007.

[49] H. Koo, Y. Chen, L. Lu, V. P. Kemerlis, and M. Polychronakis, “Compiler-assisted code randomization,” in *2018 IEEE Symposium on Security and Privacy (SP)*, 2018, pp. 461–477.

[50] T. Kremenek, P. Twohey, G. Back, A. Ng, and D. Engler, “From uncertainty to belief: Inferring the specification within,” in *7th Symposium on Operating Systems Design and Implementation*, 2006, pp. 161–176.

[51] 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*, 2012, pp. 37–48.

[52] T. Krueger, N. Krämer, and K. Rieck, “Asap: Automatic semantics-aware analysis of network payloads,” in *International Workshop on Privacy and Security Issues in Data Mining and Machine Learning*, 2010, pp. 50–63.

[53] F. R. Kschischang, B. J. Frey, and H.-A. Loeliger, “Factor graphs and the sum-product algorithm,” *IEEE Transactions on Information Theory*, vol. 47, no. 2, pp. 498–519, 2001.

[54] G. Ládi, L. Buttyán, and T. Holczer, “Message format and field semantics inference for binary protocols using recorded network traffic,” in *2018 26th International Conference on Software, Telecommunications and Computer Networks (SoftCOM)*, 2018, pp. 1–6.

[55] C. Leita, K. Mermoud, and M. Dacier, “Scriptgen: an automated script generation tool for honeyd,” in *21st Annual Computer Security Applications Conference (ACSAC)*, 2005, pp. 12–pp.

[56] X. Li and L. Chen, “A survey on methods of automatic protocol reverse engineering,” in *2011 Seventh International Conference on Computational Intelligence and Security*, 2011, pp. 685–689.

[57] Z. Lin, X. Jiang, D. Xu, and X. Zhang, “Automatic protocol format reverse engineering through context-aware monitored execution,” in *Proceedings of the Network and Distributed System Security Symposium (NDSS)*, vol. 8, 2008, pp. 1–15.

[58] Z. Lin, J. Rhee, C. Wu, X. Zhang, and X. Dongyan, “Discovering semantic data of interest from un-mappable memory with confidence,” in *Proceedings of the 19th Network and Distributed System Security Symposium (NDSS)*, vol. 12, 2012.

[59] Z. Lin, X. Zhang, and D. Xu, “Automatic reverse engineering of data structures from binary execution,” in *Proceedings of the 11th Annual Information Security Symposium*, 2010, pp. 1–1.

[60] B. Livshits, A. V. Nori, S. K. Rajamani, and A. Banerjee, “Merlin: specification inference for explicit information flow problems,” *ACM Sigplan Notices*, vol. 44, no. 6, pp. 75–86, 2009.

[61] K. Miller, Y. Kwon, Y. Sun, Z. Zhang, X. Zhang, and Z. Lin, “Probabilistic disassembly,” in *2019 IEEE/ACM 41st International Conference on Software Engineering (ICSE)*, 2019, pp. 1187–1198.

[62] D. W. Mount, *Bioinformatics: sequence and genome analysis*. Cold Spring Harbor Laboratory Press, 2004.

[63] J. Narayan, S. K. Shukla, and T. C. Clancy, “A survey of automatic protocol reverse engineering tools,” *ACM Computing Surveys (CSUR)*, vol. 48, no. 3, pp. 1–26, 2015.

[64] 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, 1970.

[65] S. Österlund, K. Razavi, H. Bos, and C. Giuffrida, “Parmesan: Sanitizer-guided greybox fuzzing,” in *29th USENIX Security Symposium (USENIX Security)*, 2020, pp. 2289–2306.

[66] R. Pang and V. Paxson, “A high-level programming environment for packet trace anonymization and transformation,” in *Proceedings of the 2003 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications*, 2003, pp. 339–351.

[67] F. Peng, Z. Deng, X. Zhang, D. Xu, Z. Lin, and Z. Su, “X-force: Force-executing binary programs for security applications,” in *23rd USENIX Security Symposium (USENIX Security)*, 2014, pp. 829–844.

[68] P. J. Phillips, A. Martin, C. L. Wilson, and M. Przybocki, “An introduction evaluating biometric systems,” *Computer*, vol. 33, no. 2, pp. 56–63, 2000.

[69] J. Pohl and A. Noack, “Automatic wireless protocol reverse engineering,” in *13th USENIX Workshop on Offensive Technologies*, 2019.

[70] J. Ren, D. J. Dubois, D. Choffnes, A. M. Mandalari, R. Kolcun, and H. Haddadi, “Information exposure from consumer iot devices: A multidimensional, network-informed measurement approach,” in *Proceedings of the Internet Measurement Conference*, 2019, pp. 267–279.

[71] A. Rosenberg and J. Hirschberg, “V-measure: A conditional entropy-based external cluster evaluation measure,” in *Proceedings of the 2007 Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning (EMNLP-CoNLL)*, 2007, pp. 410–420.

[72] M. Sharif, A. Lanzi, J. Giffin, and W. Lee, “Automatic reverse engineering of malware emulators,” in *30th IEEE Symposium on Security and Privacy (SP)*, 2009, pp. 94–109.

[73] B. D. Sija, Y.-H. Goo, K.-S. Shim, H. Hasanova, and M.-S. Kim, “A survey of automatic protocol reverse engineering approaches, methods, and tools on the inputs and outputs view,” *Security and Communication Networks*, vol. 2018, 2018.

[74] R. R. Sokal, “A statistical method for evaluating systematic relationships,” *Univ. Kansas, Sci. Bull.*, vol. 38, pp. 1409–1438, 1958.

[75] G. Starnberger, C. Kruegel, and E. Kirda, “Overbot: a botnet protocol based on kademlia,” in *Proceedings of the 4th International Conference on Security and Privacy in Communication Netowrks*, 2008, pp. 1–9.

[76] B. Stone-Gross, M. Cova, L. Cavallaro, B. Gilbert, M. Szydlowski, R. Kemmerer, C. Kruegel, and G. Vigna, “Your botnet is my botnet: analysis of a botnet takeover,” in *Proceedings of the 16th ACM Conference on Computer and Communications Security*, 2009, pp. 635–647.

[77] J. D. Thompson, D. G. Higgins, and T. J. Gibson, “Clustal w: improving the sensitivity of progressive multiple sequence alignment through sequence weighting, position-specific gap penalties and weight matrix choice,” *Nucleic Acids Research*, vol. 22, no. 22, pp. 4673–4680, 1994.

[78] R. Trimananda, J. Varmarken, A. Markopoulou, and B. Demsky, “Packet-level signatures for smart home devices,” in *27th Annual Network and Distributed System Security Symposium (NDSS)*. The Internet Society, 2020.

[79] M. von Hippel, C. Vick, S. Tripakis, and C. Nita-Rotaru, “Automated attacker synthesis for distributed protocols,” *arXiv preprint arXiv:2004.01220*, 2020.

[80] Y. Wang, X. 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 *2012 20th IEEE International Conference on Network Protocols (ICNP)*, 2012, pp. 1–10.

[81] Y. Wang, Z. Zhang, D. D. Yao, B. Qu, and L. Guo, “Inferring protocol state machine from network traces: a probabilistic approach,” in *International Conference on Applied Cryptography and Network Security*, 2011, pp. 1–18.

[82] G. Wondracek, P. M. Comparetti, C. Kruegel, E. Kirda, and S. S. S. Anna, “Automatic network protocol analysis,” in *Proceedings of the Network and Distributed System Security Symposium (NDSS)*, vol. 8, 2008, pp. 1–14.

[83] P. Wurzinger, L. Bilge, T. Holz, J. Goebel, C. Kruegel, and E. Kirda, “Automatically generating models for botnet detection,” in *European Symposium on Research in Computer Security*, 2009, pp. 232–249.

[84] Z. Xu, X. Zhang, L. Chen, K. Pei, and B. Xu, “Python probabilistic type inference with natural language support,” in *Proceedings of the 2016 24th ACM SIGSOFT International Symposium on Foundations of Software Engineering*, 2016, pp. 607–618.

[85] W. You, Z. Zhang, Y. Kwon, Y. Aafer, F. Peng, Y. Shi, C. Harmon, and X. Zhang, “Pmp: Cost-effective forced execution with probabilistic memory pre-planning,” in *2020 IEEE Symposium on Security and Privacy (SP)*, 2020, pp. 18–20.

[86] S. Zarrin and T. J. Lim, “Belief propagation on factor graphs for cooperative spectrum sensing in cognitive radio,” in *2008 3rd IEEE Symposium on New Frontiers in Dynamic Spectrum Access Networks*, 2008, pp. 1–9.

[87] Z. Zhang, W. You, G. Tao, G. Wei, Y. Kwon, and X. Zhang, “Bda: practical dependence analysis for binary executables by unbiased whole-program path sampling and per-path abstract interpretation,” *Proceedings of the ACM on Programming Languages*, vol. 3, no. OOPSLA, pp. 1–31, 2019.