Packet Field Tree: an open database, hybrid approach and evaluation measure for automated protocol reverse-engineering

Removed for anonymous review

Abstract—**Protocol specifications are useful knowledge for performing penetration testing to find network vulnerabilities, and in turn, for building intrusion-detection systems to monitor network communications. However, protocol specifications are typically verbose documents, and when new protocols are tested, considerable manual effort is required to translate these specifications into digital models of expected protocol communication. Automated protocol reverse-engineering (APRE) is a precursor to generating such digital models, and attempts to do so without any access to documentation. In this paper, we introduce *Packet Field Tree (PFT)*, the first hybrid approach combining the advantages of heuristic and supervised-learning methods. We apply it to one aspect of the APRE problem, that of discerning the boundaries between packet fields, and it does so with a superior mean *perfection*, an APRE-specific quasi-accuracy measure, of 0.27 over a variety of 21 protocols. This is an improvement of 0.11 compared to the next best APRE method. PFT also classifies the syntax of every discerned field – an output unique to our method. To measure this output we propose the *Field Tree Score*, a starting point for holistic evaluation of APRE-inferred specifications. We make the PFT and our benchmark dataset of 21 protocols available, with both designed to ease integration of community contributions. To demonstrate its practical applications, we apply the PFT to DJI drone communications and automatically discern and classify the correct field boundaries and syntaxes, corresponding to the drone's coordinates; confirming the discovery that, contrary to DJI's previous claims, this sensitive broadcasted information is *unencrypted*.**

I. Introduction

Network protocol design involves architecting structures such as inter-layer encapsulation, inter-packet state-machines, and intra-packet field formats. An ongoing question in the research of such structures is: "To what extent can each be automatically reverse-engineered from the raw bytes of the packets they generate?" Our work contributes to the inference of intra-packet structures by automatically discerning their sequences of field boundaries and classifying their respective syntaxes – the *packet format* – and in turn, the inference of all formats belonging to a protocol – the *protocol format* – solely from raw bytes in a packet capture.

ICT companies design their own proprietary protocols and keep their specifications secret, or at best publish abbreviated descriptions, to maintain a competitive advantage. Sometimes protocols are published in large documents, but not released as a machine-readable *parser*. However, parsers are often needed (i) to ensure cross-platform compatibility; (ii) to analyse malicious communications; (iii) to correct network vulnerabilities; and (iv) for archival recovery projects.

One case of growing concern is the protocol security of drones. Da-Jiang Innovations (DJI) – the world's leading civil drone manufacturer – intends its drones be used for film and photography. However, throughout the conflict in Ukraine, the DJI Mavic series has been used by both sides for reconnaissance, humanitarian aid and artillery correction [11]; thus putting the security of DJI drones under heightened scrutiny. To combat drone misuse, DJI sells its drone-and-operator-tracking *AeroScope* system to law-enforcement agencies. However, this system requires every DJI product to broadcast its location via the *DroneID* protocol – a signal that DJI had falsely claimed was encrypted and only legitimate AeroScope users could decipher [23]. Russia has been accused of exploiting this protocol to target Ukrainian civilians [24]; which was only discovered after a concerted effort to reverse-engineer the Mavic's software [6,51]. If protocol reverse-engineering (PRE) was easier, faster, more accessible – *automated* (*A*PRE) – then this flaw may have been discovered earlier, potentially saving lives.

Few APRE works include a security application absent of ground-truth discovered through other means, such as software-binary reversing in the DJI Mavic case. Shining examples of this are the Slapper Worm protocol in Autoformat [32] and Google Nest's temperature-setting packets in NetPlier [62]. By contributing to advancing the APRE field, our approach furthers the automatability of building protocol specifications of proprietary systems; which when done manually, is a slow, tedious and error-prone task.

Manually inferring protocol structures from packet captures has a long history, but automated approaches are, at present, much more limited. Such approaches involve either concurrently analysing the execution trace [10, 14, 33], fuzzing the protocol [34, 36, 45] or learning from source-code implementations [20, 53, 54]. In this work we focus on passive *network-trace* approaches to APRE. The related work (§ II) applies a variety of preprocessing, clustering, heuristic and supervised-learning techniques to infer protocol formats from the raw bytes in a packet capture – but no singular methodology demonstrates unequivocal supremacy across all contexts.

The data on which we test and compare our approach includes a greater number of protocols (21), encompassing all those previously tested by the baseline methods, and we (i) make this public for future comparisons, and (ii) engineer it in such a way that it is easy for third-party additions. In doing so, and by making our code available (§VIII), we aim to raise the state of the art in this field.

Our approach – *Packet Field Tree (PFT)* – is the first hybrid approach combining the advantages of heuristic and supervised-learning methods. We apply it firstly to one aspect of the APRE problem: that of discerning the boundaries between packet fields. PFT outperforms previous methods in the majority of cases, with a mean *perfection* – an APRE-specific quasi-accuracy measure – of 0.27. This is an improvement of at least 0.11 compared to the next best APRE approach.

Past methods have aimed at solely discerning field boundaries, but these are of limited utility by themselves. PFT innovatively classifies the field syntax, which is needed to infer field values. We introduce a new metric – *Field Tree Score* – a starting point to measure performance of the inferred specification structure. Although PFT has room for improvement, we establish benchmark scores for future APRE methods to outperform.

In summary, the contributions in this paper are:

1) An open-source database of parsed captures from 21 different protocols, compared to the previous best of releasing a sample of 9 different protocols [62].
2) The first hybrid APRE model, *Packet Field Tree*, combining heuristic and supervised methods, with a superior mean perfection score for discerning field boundaries, and the ability to classify field syntax.
3) The *Field Tree Score* for holistic evaluation of APRE-inferred specifications; where PFT scored the highest for the Modbus, TCP and EAPOL protocols.
4) The PFT automatically and correctly discerns and classifies the field boundaries and syntaxes of the coordinate fields broadcasted by the DroneID protocol.

A. *Ethical Statement*

Ethically motivated PRE is an important area of research. The goal of PRE in this and many projects is *not* to infringe on the intellectual property of any entity, but rather to provide necessary adjuncts that are either commercially unviable (such as advanced security analyses), or to allow the continuation of obsolete, unsupported, and disused protocols. When ethically applied, PRE can enhance the economic value of a protocol through extending its lifetime and ensuring its safety and security.

II. Related Work

Manual protocol reverse-engineering (PRE) has a more-than-30-year history. Frequently cited examples include reverse engineering the Server Message Block (SMB) protocol (Samba) [59], numerous chat applications [58], smart toys [47] and power-meter systems [50]. Although protocols that carry sensitive information *should* be encrypted, if a weak cipher has been cracked then the contents can certainly be reverse engineered, as shown in an analysis of avionic communications [57]. Manual PRE is very difficult, time-consuming, and is often an ongoing effort since protocols may be indefinitely updated to support new functionality [10], as evidenced by the existence of multiple SMB versions. Therefore, automated tools are required if only to aid manual solutions.

Protocols used by Internet-of-Things (IoT) devices are commonly closed-source. Hamza et al. call for IoT developers to produce a Manufacturer Usage Description (MUD) profile to aid the production of stricter firewall rules, and to verify that the network behaviour of these devices conforms to its documentation [22]. However, the MUD profile is limited to *known* protocols and endpoints used by these devices, such as from NTP or DNS requests. The exact behaviour of a custom proprietary protocol is not considered. APRE could take this security concept one step further by revealing if the behaviour of each field in a proprietary protocol conforms to some field-based profile published by the manufacturer.

In the case of DJI drones – which are also *internet-connected* – two independent works reverse engineer the DroneID protocol [6,51] with assistance from the open-source community [2,65]. In particular, these works relied on reverse engineering the drone's software to find the DroneID packet structure embedded in its code, rather than the painstaking process of manual PRE. Although there is extensive research in *execution*-trace protocol analysis, which utilises additional clues provided by the application binary, this approach requires access to the (executing) underlying applications, which may not be possible. However, *network*-trace analysis only requires passively-observed communication. The clear benefit of applying network-trace APRE in this scenario is that the invasive process of software reverse-engineering would not be necessary and physical access to the drone would not be required; which potentially would have lead to faster discovery of the security flaw in the unencypted DroneID protocol.

The sub-goals of network-trace APRE, with respect to the inference of protocol formats, include:

1) clustering similar packets within a trace [8,16,30,62],
2) discerning fields boundaries [29,30,35,62,64],
3) clustering similar fields [17,62], and
4) classifying some 'types' of field [30,35,61].

These are not unrelated tasks – they build on each other. The initial technical barrier was the inability to process misaligned field boundaries, which occur because of variable-length fields such as strings. Beddoe [5] tackled this using sequence-alignment and hierarchical-clustering algorithms, originally intended for aligning pairs of DNA sequences in the bioinformatics domain. Since Beddoe's seminal paper, multiple surveys have documented the application of other cross-domain techniques to APRE [19,25,28,38,55].

Figure 1 shows existing works organised by the technique applied, including (i) clustering methods [5,8,16,17,31,60], (ii) heuristic approaches [4,30,35,42,62] and (iii) supervised-learning techniques [61, 64]. Supervised-learning has been applied to APRE by training on a set of *known* protocols and evaluating on a testing set of *unknown* protocols [21,61,64]. The illustrated transition toward supervised-learning suggests further promise of these techniques.

The supervised-learning methods applied so far to APRE are based on fixed-packet-length inputs and fixed-field-number outputs. However, a particular packet may have variable length, contain a variable number of fields, or both. At present, the problem is addressed by either considering only protocol headers [61], pre-clustering packets using NetPlier [64], or padding [21]. To avoid the limitations of either preprocessing step, we use byte-wise features, which can be engineered for any packet length and used to discern a variable number of fields.

Supervised learning has shown tremendous ability in many fields of endeavour (*e.g.,* image classification and natural language processing), but it is driven by *very* large volumes of pre-labelled data. Obtaining large enough sets of data, and the computational load of learning across these sets, remains one of the challenges of machine learning. Hence, there are often benefits (in accuracy and reduced training sets) from incorporating some form of prior knowledge.

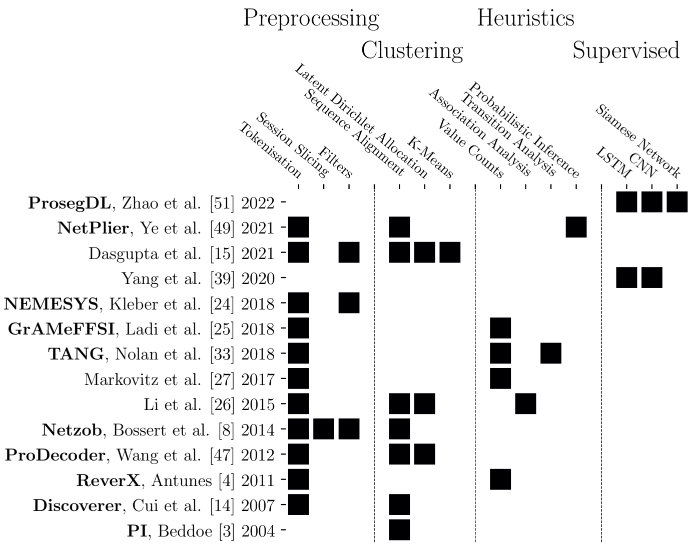

Preprocessing
Clustering
Heuristics
Supervised
Tokenisation
Session Slicing
Filters
Latent Dirichlet Allocation
Sequence Alignment
K-Means
Value Counts
Association Analysis
Transition Analysis
Probabilistic Inference
LSTM
CNN
Siamese Network
ProsegDL, Zhao et al. [51] 2022
NetPlier, Ye et al. [49] 2021
Dasgupta et al. [15] 2021
Yang et al. [39] 2020
NEMESYS, Kleber et al. [24] 2018
GrAMeFFSI, Ladi et al. [25] 2018
TANG, Nolan et al. [33] 2018
Markovitz et al. [27] 2017
Li et al. [26] 2015
Netzob, Bossert et al. [8] 2014
ProDecoder, Wang et al. [47] 2012
ReverX, Antunes [4] 2011
Discoverer, Cui et al. [14] 2007
PI, Beddoe [3] 2004

Fig. 1: **APRE Approaches:** the methods in the literature have evolved from cluster- to heuristic-based, and recently, to supervised-learning techniques. Most of the former (unsupervised) methods required some degree of tokenisation beyond considering raw bytes, such as using *n-grams* [60]. However, Yang et al. [61], Zhao et al. [64] and Garshasbi et al. [21] apply supervised-learning techniques using raw bytes as features. There exist no approaches combining heuristic and supervised techniques.

So far no method combines both heuristic and supervised-learning techniques. The advantage of heuristic methods is that they do not need extensive data sets because they encode our pre-existing knowledge of how protocols behave. However, their performance can be enhanced by incorporating the patterns mined from supervised learning. Our approach is the first to attempt a hybrid methodology that combines the advantages of both.

A. APRE Baselines

A common omission in the literature is the direct comparison of APRE tools. Although there are some 69 APRE papers surveyed in Paetzelt [43], only four [8, 27, 62, 64] of those we examined compared their work to previous approaches, and all apply their techniques to different datasets.

One of our goals is to provide a set of APRE baseline scores for comparison, but another problem is the lack of publicly-available code. Only seven tools from Paetzelt [43] provide source code. From these, we selected the three tools that have previously performed comparisons [27, 30, 62]. We use these tools as a baseline comparison and provide more detail of each below, both to understand the approach and because we use and extend ideas from Ladi et al. [30].

Nemesys: Kleber et al. [27] addressed the exponential runtime problem in sequence alignment. The authors assert that since each packet is designed to be individually parsable, one could expect indicators of the packet structure within each individual packet. They use various signal-processing filters based on the similarity of adjacent bytes to discern field boundaries on a per-packet basis.

NetPlier: Ye et al. [62] noted that alignment-based and token-based methods rely on the assumption that packets of the same format have a similar down-stream field structure, as is the case for stateless protocols. To infer stateful-protocol formats, NetPlier considers both communication directions within the network trace, and combines various heuristics in a probabilistic manner to determine the 'keyword' field.

Graph-Analysis-based Message Format and Field Semantics Inference (GrAMeFFSI): Ladi et al. [30] generated aggregate statistics for each byte index in a collection of packets. Heuristics are applied to determine whether the current byte is part of a 'static', 'enumerated' or 'highly-variable' field. This process aggregates packet formats by ultimately representing the protocol specification as a tree, whereas other APRE techniques only output an individual format for each packet. Note that our approach uses and extends this work [30].

We also use and extend the **Transition Aggregation N-Gram (TANG):** Nolan et al. [42] proposed their normalised-TANG statistic to discern the boundaries of numerical fields at the bit granularity, intended for Controller Area Network (CAN) bus protocols. TANG is constructed by counting the number of bit-flips at each index as shown in Table I.

Counter Field	Bit Index							
	7	**6**	**5**	**4**	**3**	**2**	**1**	**0**
0	0	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0	1
2	0	0	0	0	0	0	1	0
3	0	0	0	0	0	0	1	1
4	0	0	0	0	0	1	0	0
5	0	0	0	0	0	1	0	1
TANG	0	0	0	0	0	1	2	5
Norm-TANG	0	0	0	0	0	0.2	0.4	1.0

TABLE I: **Normalised-TANG Statistic:** the bits corresponding to a 1-byte counter field are stacked vertically. The bit flips (bold borders) for each position are counted in the TANG row. The counts are normalised with respect to the total number of packets. Recreated from Nolan et al. [42].

B. APRE Performance Measures

Previous works have applied evaluation measures specific to the particular APRE sub-goal investigated, or at least a qualitative assessment. This has led to a proliferation of 21 performance measures, summarised in Figure 2, where the same measure has been reused in at most 3 papers. In this paper we focus on three measures for discerning field boundaries: *perfection*, *completeness* and *conciseness*. Although there are other suitable measures, these provide equivalent information.

Although measures are aimed at evaluating sub-goals in the APRE context, an ideal solution excels at every goal. A weighted combination of measures has been proposed, but these weights lacked justification and did not account for biases in an unbalanced trace [41]. We do not apply the existing measures for the classification of field 'types', since they do not consider the whole inferred protocol structure. To address this gap, we introduce a new measure – *Field Tree Score* (FTS) – in §V-C, that focusses on the inferred structure of a protocol, rather than individual packets. This is intended as a starting point for holistic evaluation of APRE outputs: where future

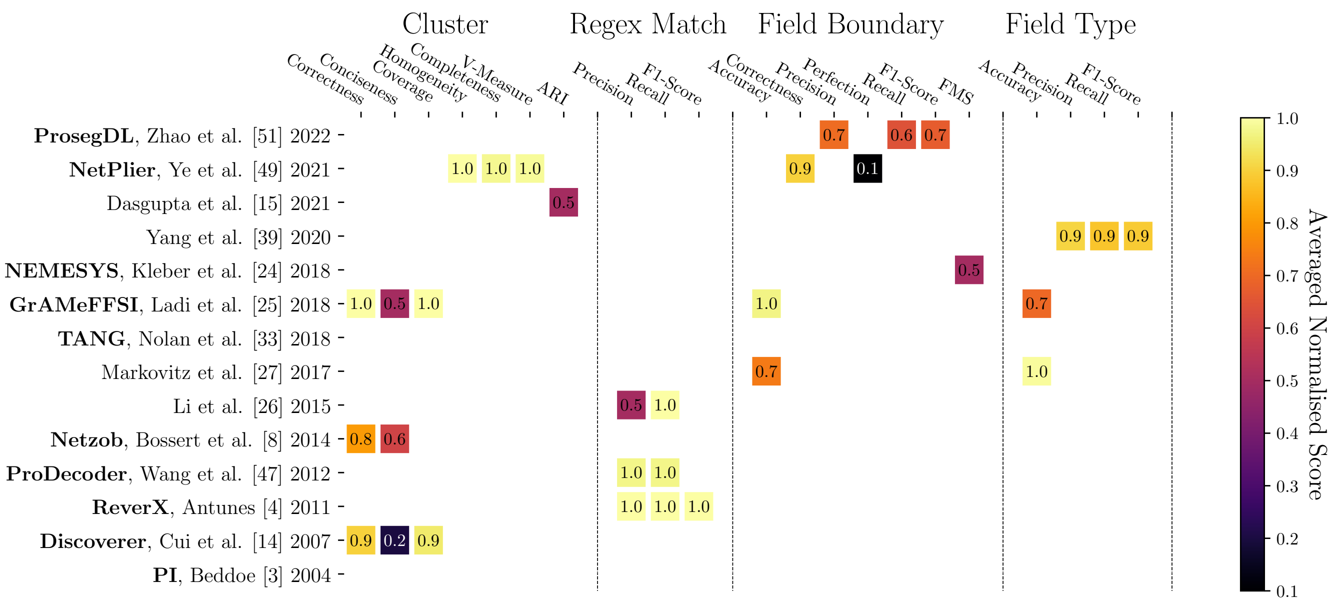


Fig. 2: **Evaluation Measures by Literature:** we show the evolution (paper date on y-axis) of scores for each measure (category on x-axis) relating to the sub-goals of APRE. Scores are normalised to a [0.0, 1.0] interval, with 1.0 the best score, and the average was calculated across the protocols tested in each paper. Note that this is far from a fair comparison since studies use different protocols of varying levels of reverse-engineering difficulty and capture size. Although [5] and [42] provide no quantitative assessment of their methods, these are highly influential works.

methods will improve in ability to classify the syntax – and perhaps *semantics* – of every discerned field.

III. An Open APRE Database

To our knowledge, no comprehensive dataset exists for the purpose of evaluating APRE techniques [62]. Although there are many publicly-available packet captures, APRE requires ground-truth field annotations. This has led to each work creating its own dataset, which has rarely been made available. This harms comparisons; for instance, most datasets having concentrated on different protocols. Across the various protocols analysed by existing works (Figure 11) – the overlaps are small. Moreover, only three works [17, 62, 64] analysed more than four protocols, and only Ye et al. [62] made their packet captures available.

We evaluate our method on 21 protocols and make our compilation of packet captures and parsed formats available (§ VIII) for validation of new methods. The protocols considered include all those previously tested by the baseline methods, including both application and non-application layers in the Open Systems Interconnection (OSI) model. Application protocols are analysed because they may share common characteristics with proprietary protocols – the main target of APRE. Non-application protocols, such as IP, TCP and UDP, usually have public specifications, and datasets are easier to obtain; thus providing more robust components for a large training set. The database traces are sourced from publicly-available repositories that are shared for general network-security research purposes [3, 12, 13, 52, 56]. When reverse engineering in the wild, we could have as little as tens or as many as thousands of packet samples, which is reflected in our dataset: ranging from 19 MQTT packets to over 82k UDP packets. Whilst textual protocols have been used in some work for APRE evaluation (Figure 11), their packet formats are not a challenging APRE target since all fields are human-readable key-value pairs. However, we include binary protocols that contain `STRING` fields, such as DNS. We make our database open-source with two intentions.

1) There is a need for an APRE benchmark; the success of such benchmark datasets has been realised in computer vision [18,49] and fuzzing [37,39], where they have raised the state of the art in their respective fields.
2) There are thousands of protocols to consider and an open-source database will allow the community to help contribute a range of APRE-relevant parsed traces.

Curating annotated traffic data manually is impractical at scale, so we use parsers specific to each protocol. But these scripts are imperfect, which presents two key challenges. Firstly, different parsing software, such as Wireshark [13] and Scapy [7], output data in different formats: Packet Description Markup Langauge (PDML) and Python data-structures, respectively. Although PDML effectively represents a parsed packet, it does not include the field syntaxes that produced the field values. Parsers for proprietary protocols, such as CAN Bus Tools [1], also add to the variety of formats in which a dissected packet could be presented. Such parsers are derived from manual PRE, which may be incomplete.

A secondary problem is that even if two parsers generate identical formats, they may still be inconsistent; for instance, because the protocol under analysis is being updated, and each parser is being developed at different rates. The protocols that are the current focus of the dataset are those for which good parsers exist. However, future work should consider protocols with less-well-developed parsers.

To avoid both problems, and remove the user requirement of downloading multiple parsing software, we include ground-truth parsed formats in our database. Hence, for new protocols to be added, users must submit a `csv` file in the correct format, along with the original packet capture for validation purposes. To reduce data repetition, two separate tables track the unique formats and the protocol-layer bytes. A sample of these respective tables, Table II and Table III, is shown for the Modbus protocol. Although not the focus of this paper, the database includes the field semantics for future works to reverse engineer.

ID	**Proto**	**Lengths**	**Syntaxes**
1	Modbus	[8,16,16]	[UINT8,UINT16,UINT16]
2	Modbus	[8,8,16]	[UINT8,UINT8,UINT16]

TABLE II: **Modbus Packet Formats Table:** there are two unique packet formats in this sample of the Modbus protocol. These formats are assigned an `ID` of 1 and 2, respectively.

Format ID	**Timestamp**	**Bytes**
1	1223541953.927963	0x0300010000
2	1223541953.929098	0x03020000
1	1223541953.929171	0x0300000001
2	1223541953.930003	0x030241c8

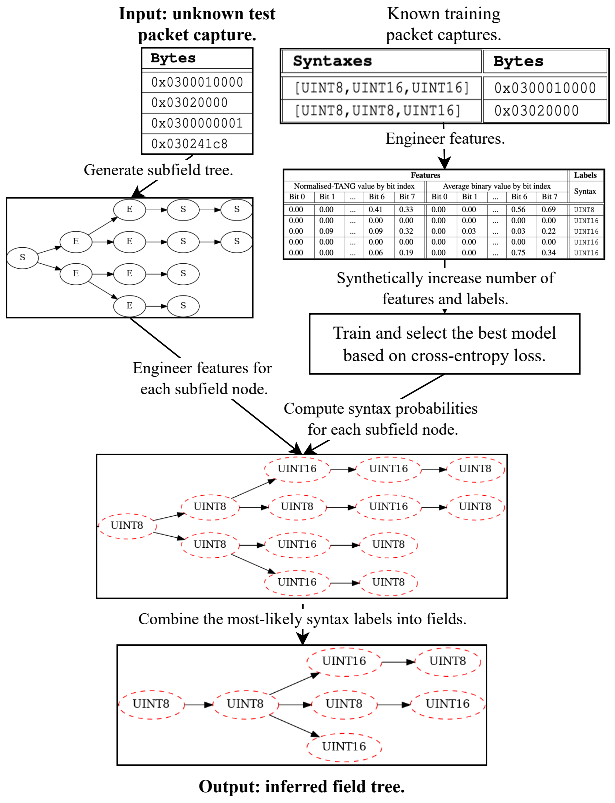
TABLE III: **Modbus Packet Bytes Table**: the 5-byte and 4-byte packets are parsed as `Format ID` 1 and 2, respectively. Since this table focusses on only the Modbus protocol, we drop the bytes from encapsulated layers.

IV. Packet Field Tree Algorithm

Our *hybrid* APRE method is summarised in Figure 3. The approach of Ladi et al. [30] shows great promise, but their GrAMeFFSI algorithm has two limitations:

1) 'Highly-variable' fields are restricted to only one byte in length and no further field-type classification is applied.
2) Once a branch contains a single remaining sequence of packet bytes, this becomes a 'constant' field and no further boundaries are discerned.

Our solution, the Packet Field Tree (PFT) algorithm, overcomes both limitations by extending GrAMeFFSI [30]. From our adaptation of the GrAMeFFSI algorithm (§IV-A), we firstly generate the subfield tree for the *unknown* protocol in the test set. Separately, a supervised model (§ IV-B) is fitted on normalised-TANG [42] and bit-averaged features, with syntax labels, from each packet format in the respective *known* training set of protocols. The best supervised model, across 4 linear or tree-based algorithms, computes a calibrated probabilistic classification of each syntax for each node in the aforementioned subfield tree of the test protocol. From these probabilities, our subfield-combination algorithm (§IV-C) discerns and classifies the most likely field boundaries and syntaxes; thus inferring the packet formats of the unknown test protocol.


Input: unknown test packet capture.
Known training packet captures.
Generate subfield tree.
E
S
Engineer features.
Synthetically increase number of features and labels.
Train and select the best model based on cross-entropy loss.
Engineer features for each subfield node.
Compute syntax probabilities for each subfield node.
UINT8
UINT16
Combine the most-likely syntax labels into fields.
Output: inferred field tree.

Bytes
0x0300010000
0x03020000
0x0300000001
0x030241c8

Syntaxes	Bytes
[UINT8,UINT16,UINT16]	0x0300010000
[UINT8,UINT8,UINT16]	0x03020000

Features										**Labels**
Normalised-TANG value by bit index					Average binary value by bit index					Syntax
Bit 0	Bit 1	...	Bit 6	Bit 7	Bit 0	Bit 1	...	Bit 6	Bit 7	
0.00	0.00	...	0.41	0.33	0.00	0.00	...	0.56	0.69	UINT8
0.00	0.00	...	0.00	0.00	0.00	0.00	...	0.00	0.00	UINT16
0.00	0.09	...	0.09	0.32	0.00	0.03	...	0.03	0.22	UINT16
0.00	0.00	...	0.00	0.00	0.00	0.00	...	0.00	0.00	UINT16
0.00	0.00	...	0.06	0.19	0.00	0.00	...	0.75	0.34	UINT16

Fig. 3: **Our PFT algorithm that adapts and extends GrAMeFFSI**: normalised-TANG-and-bit-averaged features, with syntax labels, from the packet formats in the set of known protocols are used for training, and the best supervised model selected. The subfield tree, including each node's feature array, is generated from raw bytes of the unknown test protocol. The inferred field tree is the result of combining the supervised model's probabilistic output of each syntax label from the feature array associated with each node.

A. *Subfield Tree Generation*

We now introduce the *subfield tree*, our adaptation of [30], to be extended in §IV-B and §IV-C. This tree is generated for each protocol when a member of the test set. Consider a network trace as an array $\mathbf{T}$, where $\mathbf{T}[i, j]$ is the value present in packet i at byte index j. For each byte index j, let $U(\mathbf{T}, j)$ be the number of unique values at that index in trace $\mathbf{T}$. The result of $U(\mathbf{T}, j)$ passes exactly one of three conditions, each associated with constructing a different type of node at depth j in the subfield tree.

1) If $U(\mathbf{T}, j) = 1$, *i.e.,* if only one unique value is observed at that index, then we construct a 'single' node with no siblings.
2) If $U(\mathbf{T}, j) > t$, *i.e.,* the number of unique values is greater than a given threshold t (we use the same $t = 8$ as in [30]), then also a 'single' node is constructed.
3) Else $1 < U(\mathbf{T}, j) \leq t$, and the number of unique values $U(\mathbf{T}, j)$, 'enumeration' nodes are constructed as siblings. Each unique value, $v_k : k \in [1, 2, \ldots, U(\mathbf{T}, j)]$, is

associated with each new branch. This condition partitions the trace into $U(\mathbf{T}, j)$ sub-traces. These sub-traces are formed from the packet indexes $\mathbf{i}_k$, where byte j matches v_k, *i.e.,* $\mathbf{i}_k := [i : \mathbf{T}[i,j] = v_k]$. The new sub-traces $\mathbf{T}_k$ are selected from these indicies, *i.e.,* $\mathbf{T}_k := \mathbf{T}[\mathbf{i}_k, *]$, where $*$ means across all packets. Only the respective sub-trace is considered for the next byte-index iteration for each branch.

Each time a node η is created, we retain the sequence of byte values η^B that informed this node. For example, at index j in the 'single' node case, we have $\eta^B = \mathbf{T}[*, j]$. In the 'enumeration' node case, where η_k is the node corresponding to enumeration k, $\eta_k^B = \mathbf{T}[\mathbf{i}_k, j]$. This will be needed to engineer features for each node of the subfield tree in §IV-C. We apply the three conditions left-to-right, beginning at the first byte index across the whole trace by calculating $U(\mathbf{T}, 1)$, and iterate for each byte-index, and for each sub-trace partition when the third condition passes. The result of this process is the subfield tree, where each node represents a 1-byte subfield. The idea of the 'enumeration' nodes [30], is that these will correctly partition the different packet formats and therefore each subfield will belong to only one field format.

Figure 4 shows the subfield tree generated from a short sequence of Modbus packets in Table III. The first enumeration branch, at the second byte index, partitions the packets into request formats (top two paths) or response formats (bottom two paths). Our subfield-combination algorithm in § IV-C aggregates these paths based on the outputs of the respective node features supplied to our syntax classifier in §IV-B.

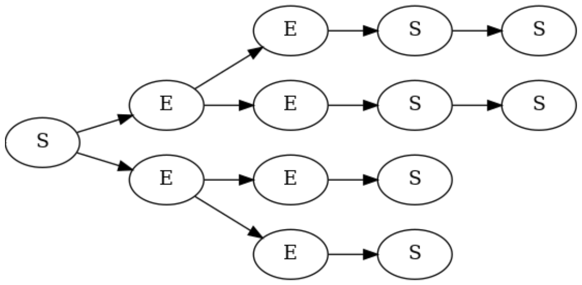


Fig. 4: **Subfield Tree for a Modbus Trace Sample:** 'enumeration' and 'single' types are labelled `E` and `S`, respectively. Each node represents a 1-byte subfield from either a 5-bytes long request, or a 4-bytes long response packet. In this example, the first byte corresponds to a constant Modbus code. The second byte, an enumeration, partitions the trace into requests (top two branches) or responses (bottom two branches).

B. Syntax Probabilistic Classifier from Subfields

The next two subsections describe our novel extensions to GrAMeFFSI. This section details our training methodology for a supervised model to compute the probability of each syntax being associated with each node in the subfield tree. Training data consists of packet captures which are partitioned into sub-traces for each unique packet format; as described in each row of the respective packet-formats table (Table II). This ensures that every byte of the same index in each sub-trace will be associated with the same syntax.

These syntaxes – the *training labels* – are adapted from the field syntaxes that Wireshark [13] assigns when parsing known protocols. We ignored syntaxes that did not provide a useful classification, *e.g.,* `BYTES` or `NONE`, or appeared in less than three of our protocols. Our resulting label set has seven syntaxes[1]:

$$Z = \{\texttt{ABSOLUTE\_TIME}, \texttt{ETHER}, \texttt{IPv4}, \texttt{STRING}, \texttt{UINT32}, \texttt{UINT16}, \texttt{UINT8}\}$$

To synthetically increase the number of packet-format sub-traces, we consider the first 16 packets, and every power of two after that, until the original subs-trace size is reached. This creates additional sub-trace samples of varying size. Sufficient model training and evaluation requires our dataset has 21 splits, each having one *unknown* protocol reserved for the test set. From the remaining 20 *known* protocols in each split, we further partition their packet-format sub-traces into a training set and validation set of 10 protocols each; ensuring each set covers all 7 syntaxes Z, and the number of sub-traces in each set is as balanced as possible. The validation set is required for model calibration and must be evenly partitioned into a calibration set and development set. Our dataset split is summarised in Table IV.

	Protocols		
	10	10	1
Packet Subseq.	Training Set	Calibration Set	Testing Set
		Development Set	

TABLE IV: **Dataset Split:** in our evaluation pipeline, each model is tested against 21 splits; each having one protocol reserved for the testing set, and the remaining 20 protocols are used to train and validate separate models. In the 10 validation protocols, half of the packet-format sub-traces are used to fit the calibrated model and the other half make up the development set.

1) Training: the sub-trace $\mathbf{T}_F$ for each packet format F is partitioned on each byte index j; forming time-ordered sequences of bytes $\mathbf{T}_F[*, j]$ for all j up to the maximum packet length. It is optimistic to think that we can discern field boundaries from individual packets [27], so we engineer bit statistics based on this ordering of bytes. For each byte sequence $\mathbf{T}_F[*, j]$ we engineer:

1) the normalised-TANG; and
2) the averaged-bit values.

By concatenating these two arrays, each *feature vector* consists of 16 floating-point values in the interval $[0, 1]$. To complete the *training sample*, a label corresponding to this byte sequence is required.

Note that most syntaxes, *e.g.,* `UINT16`, cross multiple bytes. So if a training sample is labelled as `UINT16`, this means that this byte sequence is part of a 2-byte field that together with one of its neighbours creates a `UINT16` field.

[1]`ETHER` represents a 6-byte MAC-address fields and `IPv4` represents a 4-byte IP-address field; not to be confused with their respective protocols.

Features										Labels
Normalised-TANG value by bit index					Average binary value by bit index					Syntax
Bit 0	Bit 1	...	Bit 6	Bit 7	Bit 0	Bit 1	...	Bit 6	Bit 7	
0.00	0.00	...	0.41	0.33	0.00	0.00	...	0.56	0.69	`UINT8`
0.00	0.00	...	0.00	0.00	0.00	0.00	...	0.00	0.00	`UINT16`
0.00	0.09	...	0.09	0.32	0.00	0.03	...	0.03	0.22	`UINT16`
0.00	0.00	...	0.00	0.00	0.00	0.00	...	0.00	0.00	`UINT16`
0.00	0.00	...	0.06	0.19	0.00	0.00	...	0.75	0.34	`UINT16`

TABLE V: **Modbus Training Samples:** these features are engineered from a packet subsequence matching a Modbus request format containing 5 bytes: `ID` 1 in Table II. The table shows, for a sequence of such packets, the normalised-TANG value and average bit value for each bit index, and each of the 5 rows correspond to a partition on each byte index. The packet format actually has 3 fields: a `UINT8` field, followed by a two 2-byte `UINT16` fields. However, we instead have 5 training samples since if a field crosses multiple bytes, such as `UINT16`, each feature vector engineered from the partition is paired with the same label.

Simply put, the syntax associated with the training sample is that of the (maybe larger) field to which it belongs. An example of the training samples is provided in Table V for the ModBus packet format [`UINT8`,`UINT16`,`UINT16`]. In summary, the classification process has:

1) **Features:** a vector containing 8 bit-averages, and 8 values from the normalised-TANG.
2) **Labels:** the syntax of the field obtained from Wireshark of which the respective byte sequence forms a part.

We use the training set – for each split – to separately fit four supervised models: logistic regression (LR), gradient-boosted decision trees (GBDT), random forest (RF) and linear support vector machine (LSVM) classifiers from the Scikit-learn Python library [44]. To improve the confidence in the class probability estimation of these classifiers, we employ a calibration procedure on the 10 validation protocols for each split, as detailed in the next subsection.

2) Calibration: to select which supervised model to use, we are not interested in the model with the highest classification accuracy, but rather a model that accurately reflects how confident its probabilistic classification output is. This is because we will combine subfields based on these probabilities in §IV-C. To measure each model's confidence, we use cross-entropy loss (L_{CE}):

$$L_{CE}(\mathbb{1}, \mathbb{P}) := -\frac{1}{|T|}\sum_{i=1}^{|T|}\sum_{j=1}^{|Z|} \mathbb{1}_{ij} \cdot \ln\left(\mathbb{P}_{ij}\right),$$

where $|T|$ is the number of training samples, $|Z|$ is the number of syntaxes (in our case $|Z| = 7$), $\mathbb{1}_{ij}$ (the $|T| \times |Z|$ indicator matrix) is 1 if the ith sample has the jth syntax and otherwise zero, and $\mathbb{P}_{ij}$ is probabilistic classification of the ith sample being labelled the jth syntax.

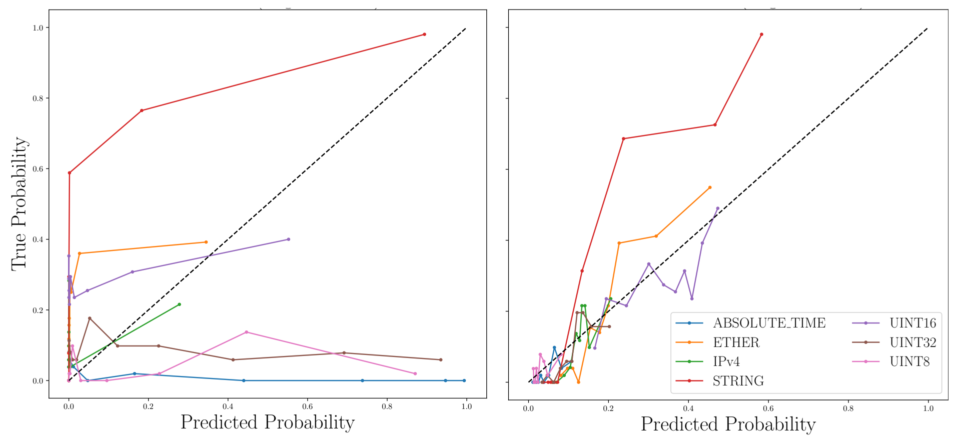
To ensure the classification probabilities accurately reflect the model's confidence, we implement calibrated classifiers [40] using Scikit-learn. This is achieved by training the weights of a sigmoid function on the calibration set, a process called Platt Scaling [46]. An isotonic regression can also be used for calibration [63]; however, this tends to overfit on small datasets such as ours [40]. Our validation set is randomly equally separated into a calibration set and development set. In Figure 5, we plot the calibration curves on the development set, for the base model before (5a) and after (5b) being calibrated on the calibration set. In the example shown, where a model has been trained on the split with Modbus reserved, the calibrated model computes probabilities that closer reflects its certainty on the development set.

We apply this classification pipeline to each protocol, *i.e.,* we repeat the whole procedure with each reserved protocol, and compute the cross-entropy loss on the development set for each model. The most successful model, with the lowest cross-entropy loss shown in bold in Table VI, is chosen for the subfield-combination algorithm in § IV-C. The cross-entropy loss was smallest (1.35) for ICMP and largest (1.71) for ARP, which are both lower than the loss for a model that computes equal probabilities for all classes (1.95). Notably, there is limited variation between model performance for a particular reserved protocol. The largest loss range was 0.17 for the MQTT protocol, which equates to a difference in the average predicted probability for the true class of 1%. Hence, our choice of the four models will have negligible impact on PFT's performance on further unseen protocols. In summary, the validation results suggest that our supervised model for each protocol is computing a reasonable syntax-probability distribution and is adequate for our subfield-combination algorithm in the next section. Further hyperparameter tuning and model exploration is left for further work, although the similar performance across models may suggest we have reached the limits of our dataset.

C. Optimal Subfield Combination

In § IV-A we generate the subfield tree from packets of an unknown protocol, and in § IV-B we trained a model to classify syntaxes from the remaining known protocols. Now we combine these results to compute the probability that each subfield node associates with each syntax, by applying the best model in Table VI to the normalised-TANG and bit-average features engineered from the byte sequences in each node of the test subfield tree η^B. From these syntax probability distributions, our subfield-combination algorithm needs to combine related subfields, *e.g.,* two adjacent nodes classified as `UINT16` should form that field together. However, note that it will be uncommon that such direct combinations are possible. More often, there will be some ambiguity such as a likely `UINT8` label followed by a likely `UINT16` label, which together could be either two `UINT8` fields or a single `UINT16` field, depending on which has the highest likelihood.

We perform this combination by applying a custom search algorithm that enforces the byte length of each syntax, *e.g.,*



(a) Base Model ($L_{CE} = 4.85$) (b) Calibrated Model ($L_{CE} = 1.38$)

Fig. 5: **Calibration Curves for Model Trained with Modbus Reserved**: these curves are calculated by binning the predicted probability of each class label into (10%) quantiles and finding the proportion of associated samples of that class. In this example, the base model is trained on 10 protocols, where Modbus is left out. We see the base model is mostly under-confident in its classifications of `STRING`, `ETHER` and `UINT16`, and mostly over-confident for the `UINT8`, `UINT32` and `ABSOLUTE_TIME` syntaxes. The calibrated model used half of the training samples, from packet subsequences in the 10 validation protocols, to reduce the cross-entropy loss by applying Platt Scaling [46]. After calibration, the computed probabilities of the resulting model is only significantly under-confident for the `STRING` syntax.

Model	Cross-entropy loss for each left-out Protocol																			
	ARP	DHCP	DNP3	EAPOL	Ethernet	ICMP	IP	Modbus	ModbusTCP	MQTT	NBDGM	NBNS	NTPv4	PMU	SMB	SMB2	S7Comm	TCP	USB	UDP
# pkts	17k	383	180	68	17k	6k	51k	202	202	19	66	2.7k	1.3k	3.7k	416	100	47 k	25k	5.5k	82k
# fmts	4	264	16	24	12	6	1	79	9	7	1	21	2	49	99	41	96	18	6	1
LR	.73	.44	.52	**.52**	.41	.39	.42	.50	.52	**.40**	.47	**.48**	.72	**.64**	.44	.56	**.65**	.54	**.49**	**.41**
GB	**.71**	.43	**.51**	.56	**.38**	**.35**	**.40**	.53	.53	.42	.44	.50	.75	.69	.43	**.51**	.70	.53	.50	.44
RF	.75	**.41**	.52	.53	.45	.39	**.40**	**.49**	**.50**	.57	**.43**	.53	.76	.73	.42	**.51**	.75	**.51**	.51	.45
SGD	.75	.46	.52	.53	.44	.42	.43	.52	.52	.49	.48	.51	**.71**	.69	**.41**	.52	.71	.54	.52	.42

TABLE VI: **Validation Results:** cross-entropy loss on the development sets for each of the trained models with respect to each reserved protocol. We include the number of packets (pkts) and number of formats (fmts) of each reserved set. For conciseness, we have **dropped the leading 1**. The best (lowest) result is bolded.

a `UINT16` field must cover exactly two bytes, to determine the most likely field sequence. However, one exception is the `STRING` syntax, which we assume can be any length, and hence our algorithm combines all adjacent nodes classified as `STRING`. A similar approach has been applied to infer individual packet formats separately [61]; but our approach considers the *whole* subfield tree to dynamically infer related formats.

To reconcile the computed syntax probabilities across the whole subfield tree – rather than individual paths – we must resolve cases where sibling nodes in the subfield tree may combine conflicting optimal syntax sequences. Consider the case where a 4-node path of the subfield tree has a high probability of `IPv4` classification and one of these nodes initiates a branch containing a 6-node path with a high probability of `ETHER` classification. We can only pick one of these conflicting classifications. We resolve such cases by computing the most likely syntax labelling for every subtree of the subfield tree, performing recursive upward traversal to the root node, and choosing its determination of the optimal fields. To help explain this process, we define:

$$\begin{aligned} C(\eta) &:= \text{Set of children nodes relative to node } \eta, \\ H_{-1} &:= \text{The last node in path } H, \\ D(\eta, m) &:= \text{Set of } m\text{-node descending paths from node } \eta \text{ (inc.)}, \\ S(\eta) &:= \text{The syntax associated with } \eta, \text{ initially } \texttt{NONE}, \text{ and} \\ P(\eta, \zeta) &:= \text{Probability that node } \eta \text{ has syntax } \zeta. \end{aligned}$$

In this recursive algorithm, the base case considers a leaf node η_1. In isolation, the only possible syntax of 1-byte length is `UINT8` and so $S(\eta_1) := \texttt{UINT8}$.

For the second node case η_2, η_1's parent, it in isolation with η_1 must be classified a 2-byte-length syntax or be two `UINT8` nodes. To classify which case, we compare:

$$\mathbb{P}(\eta_2 \text{ is } \texttt{UINT16}) := \prod_{H \in D(\eta_2, 2)} \left(\prod_{\eta_h \in H} P(\eta_h, \texttt{UINT16}) \right)$$

with

$$\mathbb{P}(\eta_2 \text{ is } \texttt{UINT8}) := P(\eta_2, \texttt{UINT8}) \times \prod_{\eta_c \in C(\eta_2)} P(\eta_c, \texttt{UINT8}).$$

If $\mathbb{P}(\eta_2 \text{ is } \texttt{UINT16}) > \mathbb{P}(\eta_2 \text{ is } \texttt{UINT8})$, then $S(\eta_2) := \texttt{UINT16}$, otherwise $S(\eta_2) := \texttt{UINT8}$. We compute similar products for η_3; but for η_4, the `UINT32` and `IPv4` syntaxes must also be considered.

In the general case, once we are early enough in the tree such that the current node η_g could mark the head of any syntax, we compare the product of probabilities for each syntax ζ and its length l. For each case, we compute $\mathbb{P}(\eta_g \text{ is } \zeta) :=$

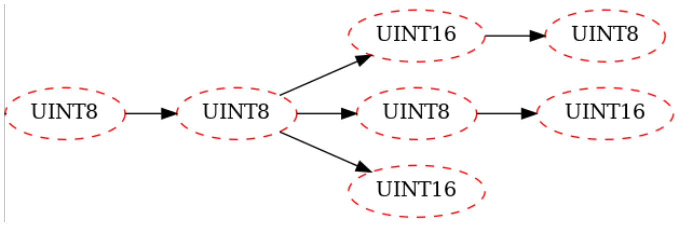
$$\prod_{H \in D(\eta_g, l)} \left(\left(\prod_{\eta_h \in H} P(\eta_h, \zeta) \right) \times \prod_{\eta_c \in C(H_{-1})} P(\eta_c, S(\eta_c)) \right),$$

the product of the probability that all the descending nodes from η_g up to l hops away are classified as syntax ζ, and the previously-calculated probability of all the $(l+1)$-hop-away nodes marking the head of some syntax. For the syntax where $\mathbb{P}(\eta_g \text{ is } \zeta)$ is highest, then $S(\eta_g) := \zeta$. This process continues until we reach the root node η_r.

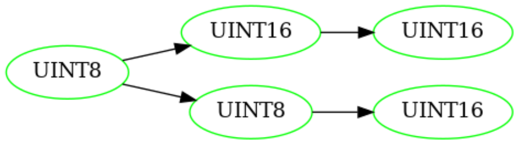
To infer the whole protocol format, we unpack the associated syntaxes starting from η_r. Hence, the first syntax classification is $S(\eta_r)$ and its respective length l_r. Next, the second field classifications branch to the unique syntaxes from

$$\{S(\eta_c) \text{ for } \eta_c \in C(H_{-1}) \text{ for } H \in D(\eta_r, l_r)\},$$

the set of syntaxes of the l_r-hop-away (from the root node η_r) child nodes. We continue rightward to the leaf nodes in this fashion, where the combination process guarantees the length of the terminal syntax classification matches the number of remaining subfield nodes. The result of this process is the *inferred field tree*, and we refer to the ground-truth field-structure as the *true field tree* (TFT), as shown in Figure 6. Now instead of four branches generated by the subfield tree in § IV-A, these have been combined into the two formats corresponding to either request or response packets. However, there remain incorrect syntax classifications; which we evaluate in §V.



(a) Inferred Field Tree for the Modbus packet sample.



(b) True Field Tree (TFT) for the Modbus packet sample.

Fig. 6: **PFT for Modbus Sample:** the combination process results in the inferred field tree (red dashed). Here the bottom branch of each tree match exactly. The top branch of the TFT (green solid) maps to two different inferred branches; both match the first `UINT8` field and one matches the last `UINT16` field.

V. Evaluation

Our evaluation of the PFT, and the three baseline comparisons, is three-fold:

1) How does our method perform with respect to discerning field boundaries?
2) What properties of the protocols studied influence the quality of results?
3) How does the PFT's syntax classification and structure inference perform?

We report results for each protocol. In each case, experiments are conducted leaving this protocol out of the training and validation sets.

A. Boundary Evaluation

We focus on the same field-boundary *perfection* measure used by Ye et al. [62]:

$$\textbf{Perfection:} \ \frac{\text{\# of discerned fields that match a true field}}{\text{\# of true fields}}.$$

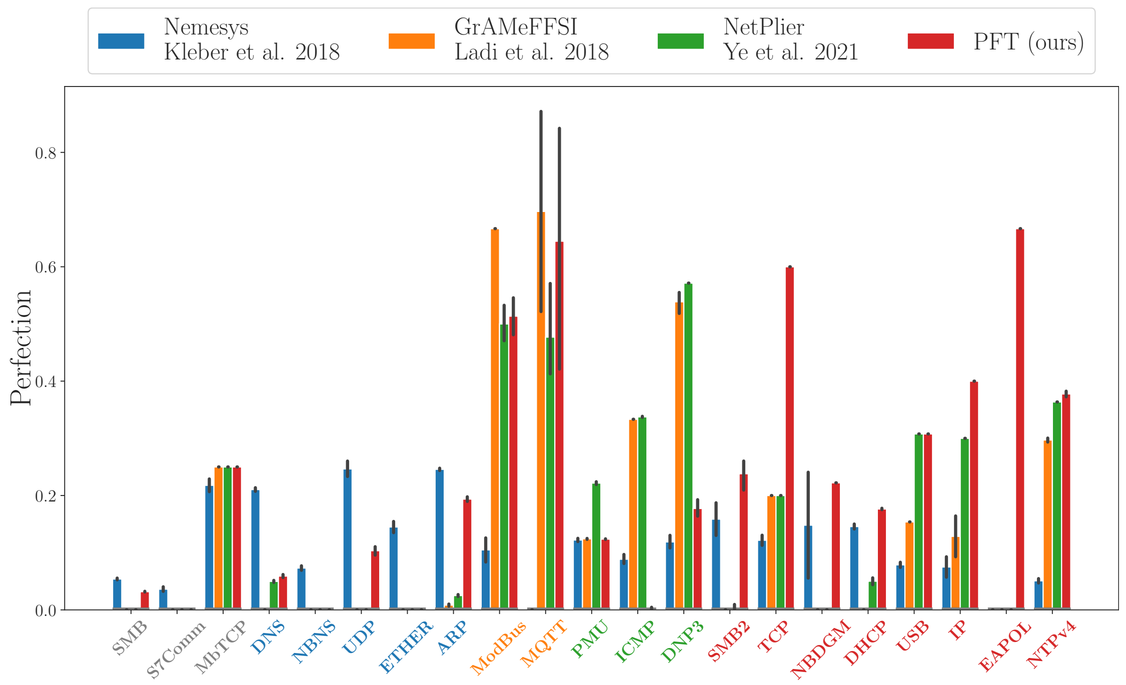
In addition, rather than using their *correctness* measure, we separate this into *completeness* and *conciseness* to verify if the model is discerning more or less fields than necessary, respectively:

$$\textbf{Completeness:} \ \frac{\text{\# of discerned fields within a true field}}{\text{\# of discerned fields}},$$
$$\textbf{Conciseness:} \ \frac{\text{\# of true fields within a discerned field}}{\text{\# of true fields}}.$$

Across the 21 protocols in our dataset, we calculate each measure for the PFT and the three baseline methods. Figure 7 shows that PFT provided the highest perfection for 7 protocols and equal highest for a further 2 protocols. Our mean and median perfection scores were higher than the next best APRE model by 0.11 and 0.10, respectively. In the worst case, GrAMeFFSI and NetPlier had perfection scores of 0.0 for 11 and 8 protocols, respectively. However, there are some cases where PFT performs worse than at least one of the alternatives. This reflects the difficulty of creating a universal protocol-format inference engine. The scores can no doubt be improved (ideally 1.0), but this represents an almost doubled performance over the best alternatives. Also, when we examine completeness and conciseness measures, PFT is not so outstanding. It is hence worth understanding why and when it performs well, and why perfection is an imperfect measure.

B. Protocol Properties

Although PFT provides the highest perfection for the majority of protocols, we would like to understand what factors contribute to the performance of each method. We consider two simple, compositional, protocol trace properties: (i) the proportion of fields that are one byte in length, and (ii) the proportion of fields that only take on a single value in the trace. We show the average behaviour of each protocol on



Model	Perfection		Completeness		Conciseness	
	mean	median	mean	median	mean	median
Nemesys	0.12	0.12	0.52	0.58	**0.74**	0.75
GrAMeFFSI	0.16	0.01	0.68	0.72	0.70	**0.80**
NetPlier	0.16	0.05	**0.85**	**0.94**	0.51	0.50
PFT (ours)	**0.27**	**0.22**	0.80	0.83	0.62	0.71

Fig. 7: **Perfection, Completeness and Conciseness comparisons.** The x-axis colour indicates which model scored the highest for that particular protocol, where grey means there was a difference of less than 0.05 across all 4 models. The error bars capture the range of scores due to the variation in packet formats for each protocol. We find that the results are protocol dependent, and for all models, there are cases where no field boundaries are correctly discerned. Given this variability, we calculate both the mean and median for each measure. Our PFT method, had the highest mean and median perfection. In the completeness calculations, NetPlier had the highest mean and median, suggesting tendency to discern too many fields. Regarding the conciseness calculations, Nemesys and GrAMeFFSI had the highest mean and median, respectively, suggesting tendency to discern too few fields. In the perfection calculations, our model has the highest mean and median over 21 different protocols, **0.27** and **0.22**, respectively.

a scatter plot in Figure 8. We find clear regions where each model performs the best:

- Nemesys has the most success on protocols with a low proportion of `UINT8` fields. It under-discerns fields (high conciseness) at the expense of missing boundaries. Thus, it is not good on protocols with many small fields.
- GrAMeFFSI performs the best on protocols with a high proportion of `UINT8` fields and low proportion of single-value fields, because it relies heavily on the statistical information carried in multi-value fields.
- NetPlier was the highest performing model for protocols with the average range of `UINT8` proportion, probably because of its tendency to over-infer fields, which we see in its high completeness results.
- PFT performed the best in the area that is difficult for other approaches, with both medium-to-high proportion of `UINT8` fields and moderately high number of single value fields. But PFT's performance was often the second best in other regions such as protocols with a low portion of `UINT8` fields.

It is unsurprising that all models had poor performance on the SMB and S7Comm packet sequences, since they had the *highest* proportion of single-value fields; which makes discerning field boundaries difficult because the majority of packets in the dataset appear static (constant).

Note that these results illustrate well the need for a better dataset and baseline comparisons. The biases in GrAMeFFSI possibly arose because it was solely tested on MQTT and Modbus in the original paper (Figure 11). Likewise, Nemesys was evaluated on DHCP, DNS and NTP; all of which have less than half of their fields labelled as `UINT8`.

Although perfection is a useful measure to allow comparisons with existing APRE tools, this measure is not ideal when applied to unbalanced traces. Since perfection weights all packets equally, if a model scored highly for one packet format that covers a large proportion of the trace, the model will still receive a high score if it *imperfectly* infers the remaining

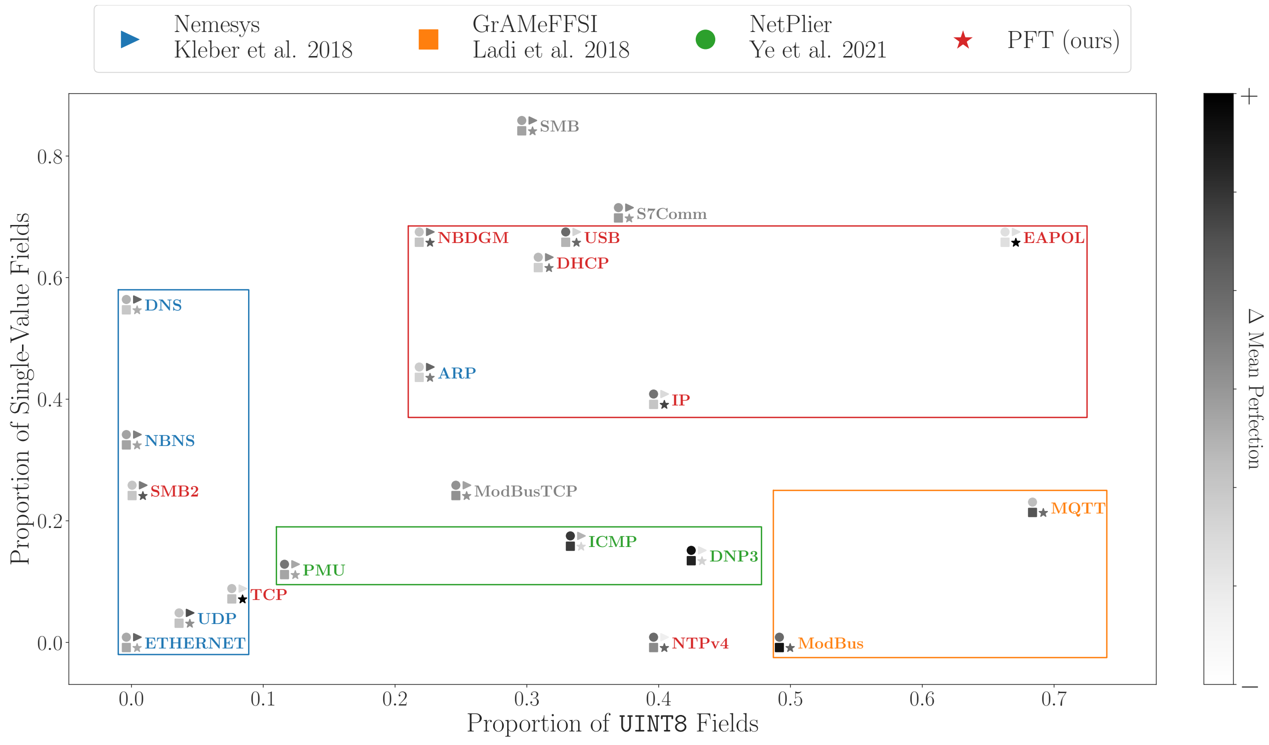


Fig. 8: **Relative Model Performance with respect to Protocol Properties:** each protocol is coloured in accordance with the model that has the highest perfection score for that protocol, indicated by the positive (black) delta with respect to the mean perfection across all four methods. We plot rough regions where the same model had the highest performance across protocols in that region. SMB, S7Comm and ModbusTCP are greyed-out because no model significantly outperformed any other.To avoid overlapping text, an offset has been added to UDP, TCP and PMU, which would have also been located at $(0, 0)$ with Ethernet.

packet formats. This realisation motivates our new method of syntax evaluation, in which APRE is evaluated against the protocol specification as a whole rather than an aggregation of individual packet formats.

C. Field Structure Evaluation

Fixed headers, *i.e.,* a single packet format, are common in lower layers of the protocol stack, such as Ethernet, IP and UDP, and so § V-A explained the common perfection-metric approach to evaluate the inferred packet formats. However, application-layer protocols typically contain multiple packet formats with inter-field dependencies. APRE research lacks a measure that evaluates a model's ability to infer the *field structure*, *i.e.,* the parsing logic, of the unknown protocol specification. The motivation for evaluating the inferred field structure is that the parsing logic is required to develop consequent tools, such as real-time packet processing [9], network intrusion-detection systems [48] and protocol fuzzers [45], which may be exposed to new packet bytes generated by the unknown protocol – unseen during the model fitting. Simply outputting packet formats on a per-packet basis still puts effort on the security analyst to infer the parsing logic. Certainly existing APRE solutions could store a dictionary of learned packet formats and feasibly parse these during real-time inference, however existing solutions cannot infer formats from unseen packet bytes within typical inter-arrival times, which may be as short as 0.5ms in a CAN Bus network [26]. Our PFT algorithm, presents such parsing logic in the 'enumeration' branches that are merged together by our subfield-combination algorithm. Hence, we propose the *Field Tree Score* (FTS) to provide an evaluation of our tree representation of a protocol specification as a starting point for holistic field-structure evaluation.

The FTS measure is similar to the tree edit distance (TED) without relabelling. A node is considered matched if the classified and true syntax (the node label) are the same. If they are different, the classified node must be deleted and the true syntax label added. The key difference to TED is that rather than adding the costs of insertions and deletions to get a distance, we normalise the number of insertions and deletions to get a score such that 1 is a perfect match and 0 is a complete mismatch. To calculate the FTS, we use first use TED to count:

PN : the number of nodes in the PFT;
TN : the number of nodes in the TFT;
PND : the number of nodes that need to be deleted; and
TNI : the number of nodes that need to be inserted.

We combine these four values such that a score of 0 or 1, means the whole inferred structure is incorrect or correct, respectively:

$$\textbf{FTS} := 1 - \omega\frac{\text{PND}}{\text{PN}} - (1-\omega)\frac{\text{TNI}}{\text{TN}},$$

where $0 \leq \omega \leq 1$. In our results, we chose $\omega := 0.5$ to equally weight the cost of an end user – a security practitioner

	Protocol																				
	ARP	DHCP	DNP3	DNS	EAPOL	Ethernet	ICMP	IP	Modbus	ModbusTCP	MQTT	NBDGM	NBNS	NTPv4	PMU	SMB	SMB2	S7Comm	TCP	USB	UDP
PN	63	48	8	29	4	12	18	34	7	4	28	45	23	437	284	20	65	32	4	21	11
TN	9	22	7	9	3	3	6	10	5	4	14	9	9	11	8	30	33	7	5	13	4
PND	60	36	5	27	2	9	16	30	2	3	22	43	23	431	283	11	54	30	1	15	10
TNI	6	11	4	7	1	0	4	6	0	3	8	7	9	5	7	21	22	5	2	7	3
FTS	.19	.38	.41	.15	**.58**	.24	.22	.26	.86	.25	.32	.13	.00	.28	.06	.38	.25	.17	**.68**	.37	.13

TABLE VII: **PFT Graph-Based Evaluation:** to compare how well the inferred PFT matches the TFT, we use our FTS ($\omega_1 = 0.5$). None of the node classifications were correct for NBNS, PMU or UDP. The PFT discerned a great number of nodes for NTPv4 and PMU, which suggests that there were many 'enumeration' nodes in the subfield tree that failed to be combined in our 'optimal field combination' (§IV-C). The PFT scored the highest for Modbus, TCP and EAPOL, with scores of **0.86**, **0.68** and **0.58**, respectively.

– inserting and/or deleting nodes to match the TFT; determined from manual PRE or known documentation. However, our measure allows for non-equal weighting if such users agree it's easier to either add or delete formats.

Continuing with the Modbus example, as shown in Figure 6, we have 7 discerned nodes (PN $:= 7$) and 5 true nodes (TN $:= 5$). In 6a, the `UINT8` root node and the adjacent `UINT8` node both match 6b, so do not need to be removed. However, the remaining `UINT32` node and two `UINT8` nodes do not match and must be deleted, hence PND $:= 2$. We do not need to insert any nodes to produce the TFT in 6b, so TNI $:= 0$. Therefore, we have:

$$\textbf{FTS}_{\text{Modbus}} := 1 - (0.5)\frac{2}{7} - (0.5)\frac{0}{5} = 0.86$$

In Table VII, we report the FTS for each protocol. In summary, PFT scored the highest for TCP and EAPOL, with 0.68 and 0.58, respectively. However, these are simplistic protocols as indicated by the number of true nodes: 5 and 3, respectively. Other scores are somewhat variable. Note that we are unable to compare these scores to the baseline models as they do not classify field syntax, which is a strong contribution of our model. We encourage the development of new APRE syntax-classification techniques to directly improve on our FTS results.

VI. Application to DJI DroneID Protocol

To demonstrate the security implications of successful protocol reverse-engineering, we show how the PFT can discern the fields and correctly classify syntaxes corresponding to the latitude and longitude broadcasted by a DJI drone. With permission from [15], we use their packet capture of simulated DroneID communications and to compare our inference with the ground truth, we use their Wireshark dissector. In the DroneID protocol, the drone's and the operator's current coordinates are sent as radians; converted to a `UINT32` by multiplying the values by 10^7. We apply our PFT algorithm (with the trained model with lowest L_{CE}) using the whole DroneID packet capture as input, and in Figure 9 we show the inferred branches for just the fields leading up to each set of coordinates. Although PFT cannot automatically label these fields with some `COORDINATE` semantic, only tens of inferred nodes need to be analysed, rather than hundreds of packets each with hundreds of bytes. To demonstrate the security implications of this information, we plot the drone coordinates in Figure 10 – APRE has rapidly discovered this sensitive broadcast without needing to reverse engineer the Mavic's software. By contributing to advancing the APRE field, PFT furthers the automatability of building protocol specifications of proprietary systems, which can take days to do manually [47,50,58,59].

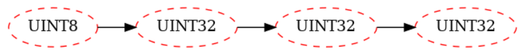
VII. Limitations & Further Work

In §IV-B, we stated that further hyperparameter tuning was unnecessary. However, a small improvement in the 1-byte classification model could lead to unpredictable improvements in the evaluation results. Deep-learning and sequence-based models could potentially achieve such an improvement.

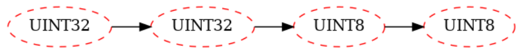
Wireshark not only provides syntax classification, but also crucially includes each field's semantic information. Our PFT approach could be applied to inform field semantics, such as `Length`, `Counter` and `Address`, in addition to field syntaxes. Although the semantic information is included in our database, some natural language processing is required to transform these into suitable labels.

Our approach is limited to fields at the byte granularity. Many proprietary protocols, such as CAN Bus, operate using sub-byte fields since data must be tightly packed to minimise latency. Further work should explore approaches that effectively infer all-sized fields: from 1-bit flags to `STRING` fields potentially spanning hundreds of bytes.

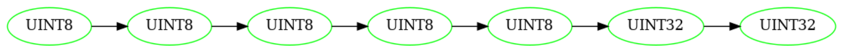
Our approach assumes that protocols can be concisely represented as an acyclic tree of fields, and that enumeration branches are sufficient to differentiate between packet formats. However, some protocols may be better represented by a cyclic graph of fields; such as recurring DNS `Resource Record` fields. Similarly, our approach cannot indicate hierarchical fields (`Resource Record` fields are also an example of this), which are indicated when known protocols are parsed in Wireshark.



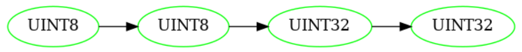
(a) Inferred Field Tree for **Drone** Coordinates.



(b) Inferred Field Tree for **Operator** Coordinates.



(c) TFT for **Drone** Coordinates (last two fields).



(d) TFT for **Operator** Coordinates (last two fields).

Fig. 9: **DJI Drone and Operator Coordinate Packet Inference:** the PFT algorithm correctly discerned and classified the last two `UINT32` fields; corresponding to the latitude and longitude coordinate of the drone. However, the PFT failed to do so for the operator coordinates; likely since these were static in the capture and no variation in the last 8 byte values was observed.

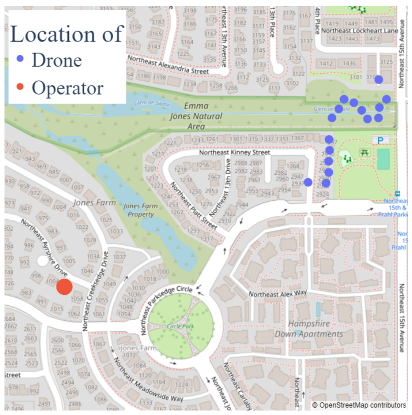


Fig. 10: **Drone and operator tracking proof of concept**: by plotting the latitude and longitude values as points, we can determine the path of the drone, and the operator's location.

Whilst our APRE benchmark dataset currently has captures and dissections of 21 protocols, a much larger number is required to measure and understand each model's ability to generalise across a range of protocol properties. This need is emphasised by the further application of data-driven methods, such as supervised-learning, that improve with higher quality and larger volumes of data.

VIII. Conclusion

In this paper, we introduced the Packet Field Tree (PFT) and provided comparisons with existing models for 21 different protocols. The PFT is the first hybrid approach, combining heuristic and supervised methods, to one aspect of the APRE problem concerning packet formats. Our model offered equal or higher perfection for 9 protocols, compared to three baseline models, and had the highest perfection mean of 0.27. Our model also had the highest performance on a wider range of protocols in our dataset, with respect to their proportion of 1-byte-length fields and single-value fields. In addition, the PFT classifies the field syntax and wholistically infers the protocol format. We proposed the Field Tree Score to fairly measure the success of this addition, and encourage its application to the evaluation of future APRE methods. In particular, we applied the PFT to DJI drone communications and rapidly and correctly discerned and classified the field boundaries and syntaxes, corresponding to the drone's coordinates. APRE may have discovered this sensitive broadcast earlier, potentially saving lives.

Availability

We make our APRE benchmark dataset, Packet Field Tree code and the Field Tree Score code available.[2]

References

[1] "CAN bus tools in Python 3," https://github.com/cantools.

[2] "DJI-Firmware-Tools: Tools for handling firmwares of DJI products, with focus on quadcopters." https://github.com/o-gs/dji-firmware-tools.

[3] 4SICS, "Geek Lounge ICS Lab (CS3STHLM)," download.netresec.com/pcap/4sics-2015, 2015.

[4] J. Antunes, N. Neves, and P. Verissimo, "Reverse Engineering of Protocols from Network Traces," in *2011 18th Working Conference on Reverse Engineering*. Limerick, Ireland: IEEE, Oct. 2011, pp. 169–178.

[5] M. A. Beddoe, "Network Protocol Analysis using Bioinformatics Algorithms," McAfee, https://github.com/bitpeach/Protocol-Informatics, Tech. Rep., 2004.

[6] C. Bender and J. Staggs, "Leveling the playing field: Equipping Ukrainian freedom fighters with low-cost drone detection capabilities," in *2023 15th International Conference on Cyber Conflict: Meeting Reality (CyCon)*. Tallinn, Estonia: IEEE, May 2023, pp. 287–312. [Online]. Available: https://ieeexplore.ieee.org/document/10181421/

[7] P. Biondi, "Packet generation and network based attacks with Scapy," Suresnes, FRANCE, 2005.

[8] G. Bossert, F. Guihéry, and G. Hiet, "Towards automated protocol reverse engineering using semantic information," in *9th ACM Symposium on Information, Computer and Communications Security (ASIA CCS)*. Kyoto Japan: ACM, Jun. 2014, pp. 51–62.

[9] P. Bosshart, D. Daly, G. Gibb, M. Izzard, N. McKeown, J. Rexford, C. Schlesinger, D. Talayco, A. Vahdat, G. Varghese, and D. Walker, "P4: Programming protocol-independent packet processors," *SIGCOMM Comput. Commun. Rev.*, vol. 44, no. 3, pp. 87–95, Jul. 2014.

[10] J. Caballero and D. Song, "Automatic protocol reverse-engineering: Message format extraction and field semantics inference," *Computer Networks*, vol. 57, no. 2, pp. 451–474, Feb. 2013.

[11] A. Chapple, "The drones of the Ukraine war," *Radio Free Europe/Radio Liberty*, 2022.

[12] CISA, "Industrial Control Systems Network Protocol Parsers (ICSNPP)," github.com/cisagov/ICSNPP, 2020.

[13] G. Combs, "Wireshark," wiki.wireshark.org, 1998.

[14] P. M. Comparetti, G. Wondracek, C. Kruegel, and E. Kirda, "Prospex: Protocol Specification Extraction," in *2009 30th IEEE Symposium on Security and Privacy*. Oakland, CA, USA: IEEE, May 2009, pp. 110–125.

[2]https://anonymous.4open.science/r/apre-benchmark-database-p47/README.md

[15] G. Cox, "Open DroneID Wireshark dissector," https://github.com/opendroneid/wireshark-dissector.

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

[17] A. Dasgupta, Y.-X. Yan, C. Ong, J.-Y. Teo, Bugsy, C.-W. Lim, and Andrew, "Exploring Unsupervised Learning Methods for Automated Protocol Analysis," in *2021 IEEE Symposium Series on Computational Intelligence (SSCI)*, 2021, pp. 1–8.

[18] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei, "Imagenet: A large-scale hierarchical image database," in *2009 IEEE conference on computer vision and pattern recognition*. IEEE, 2009, pp. 248–255.

[19] J. Duchêne, 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, Feb. 2018.

[20] T. Ferreira, H. Brewton, L. D'Antoni, and A. Silva, "Prognosis: Closed-Box Analysis of Network Protocol Implementations," in *Proceedings of the 2021 ACM SIGCOMM 2021 Conference*, Aug. 2021, pp. 762–774, arXiv:2201.02577 [cs]. [Online]. Available: http://arxiv.org/abs/2201.02577

[21] J. Garshasbi and M. Teimouri, "CNNPRE: A CNN-based protocol reverse engineering method," *IEEE Access*, vol. 11, pp. 116 255–116 268, 2023. [Online]. Available: https://ieeexplore.ieee.org/document/10287339/

[22] A. Hamza, D. Ranathunga, H. H. Gharakheili, T. A. Benson, M. Roughan, and V. Sivaraman, "Verifying and Monitoring IoTs Network Behavior Using MUD Profiles," *IEEE Transactions on Dependable and Secure Computing*, vol. 19, no. 1, pp. 1–18, Jan. 2022.

[23] S. Hollister, "DJI drones, Ukraine, and Russia — what we know about AeroScope," https://www.theverge.com/22985101/dji-aeroscope-ukraine-russia-drone-tracking, Mar. 2022.

[24] ——, "DJI insisted drone-tracking AeroScope signals were encrypted — now it admits they aren't," https://www.theverge.com/2022/4/28/23046916/dji-aeroscope-signals-not-encrypted-drone-tracking, Apr. 2022.

[25] Y. Huang, H. Shu, F. Kang, and Y. Guang, "Protocol Reverse-Engineering Methods and Tools: A Survey," *Computer Communications*, vol. 182, pp. 238–254, Jan. 2022.

[26] S. Jeong, S. Lee, H. Lee, and H. K. Kim, "X-canids: Signal-aware explainable intrusion detection system for controller area network-based in-vehicle network," *IEEE Transactions on Vehicular Technology*, vol. 73, no. 3, pp. 3230–3246, 2024.

[27] 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)*. Baltimore, MD: USENIX Association, Aug. 2018.

[28] S. Kleber, L. Maile, and F. Kargl, "Survey of Protocol Reverse Engineering Algorithms: Decomposition of Tools for Static Traffic Analysis," *IEEE Communications Surveys & Tutorials*, vol. 21, no. 1, pp. 526–561, 2019.

[29] S. Kleber, R. W. van der Heijden, and F. Kargl, "Message Type Identification of Binary Network Protocols using Continuous Segment Similarity," *arXiv:2002.03391 [cs]*, Feb. 2020.

[30] G. Ladi, L. Buttyan, and T. Holczer, "Message Format and Field Semantics Inference for Binary Protocols Using Recorded Network Traffic," in *26th International Conference on Software, Telecommunications and Computer Networks (SoftCOM)*. Split: IEEE, Sep. 2018, pp. 1–6.

[31] H. Li, B. Shuai, J. Wang, and C. Tang, "Protocol Reverse Engineering Using LDA and Association Analysis," in *2015 11th International Conference on Computational Intelligence and Security (CIS)*. Shenzhen, China: IEEE, Dec. 2015, pp. 312–316.

[32] Z. Lin, X. Jiang, D. Xu, and X. Zhang, "Automatic protocol format reverse engineering through context-aware monitored execution," in *Network and Distributed System Security Symposium*, 2008. [Online]. Available: https://api.semanticscholar.org/CorpusID:6611986

[33] ——, "Automatic protocol format reverse engineering through context-aware monitored execution," in *In Network and Distributed System Security*, 2008.

[34] Z. Luo, K. Liang, Y. Zhao, F. Wu, J. Yu, H. Shi, and Y. Jiang, "DynPRE: Protocol reverse engineering via dynamic inference," in *Proceedings 2024 Network and Distributed System Security Symposium*. San Diego, CA, USA: Internet Society, 2024. [Online]. Available: https://www.ndss-symposium.org/wp-content/uploads/2024-83-paper.pdf

[35] M. Markovitz and A. Wool, "Field classification, modeling and anomaly detection in unknown CAN bus networks," *Vehicular Communications*, vol. 9, pp. 43–52, Jul. 2017.

[36] R. Meng, M. Mirchev, M. Böhme, and A. Roychoudhury, "Large Language Model guided Protocol Fuzzing," in *Proceedings 2024 Network and Distributed System Security Symposium*. San Diego, CA, USA: Internet Society, 2024. [Online]. Available: https://www.ndss-symposium.org/wp-content/uploads/2024-556-paper.pdf

[37] J. Metzman, L. Szekeres, L. Simon, R. Sprabery, and A. Arya, "FuzzBench: An open fuzzer benchmarking platform and service," in *Proceedings of the 29th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*. Athens Greece: ACM, Aug. 2021, pp. 1393–1403.

[38] J. Narayan, S. K. Shukla, and T. C. Clancy, "A Survey of Automatic Protocol Reverse Engineering Tools," *ACM Computing Surveys*, vol. 48, no. 3, pp. 1–26, Feb. 2016.

[39] R. Natella and V.-T. Pham, "ProFuzzBench: A benchmark for stateful protocol fuzzing," in *Proceedings of the 30th ACM SIGSOFT International Symposium on Software Testing and Analysis*. Virtual Denmark: ACM, Jul. 2021, pp. 662–665.

[40] A. Niculescu-Mizil and R. Caruana, "Predicting good probabilities with supervised learning," in *22nd International Conference on Machine Learning (ICML)*. Bonn, Germany: ACM Press, 2005, pp. 625–632.

[41] B. Ning, X. Zong, K. He, and L. Lian, "PREIUD: An Industrial Control Protocols Reverse Engineering Tool Based on Unsupervised Learning and Deep Neural Network Methods," *Symmetry*, vol. 15, no. 3, p. 706, Mar. 2023.

[42] B. C. Nolan, S. Graham, B. Mullins, and C. S. Kabban, "Unsupervised time series extraction from controller area network payloads," in *88th IEEE Vehicular Technology Conference (VTC-Fall)*, 2018, pp. 1–5.

[43] A. Paetzelt, "List of (automatic) protocol reverse engineering tools/methods/approaches for network protocols," github.com/techge/PRE-list, github.com/techge/PRE-list, 2021.

[44] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay, "Scikit-learn: Machine learning in Python," *Journal of Machine Learning Research*, vol. 12, pp. 2825–2830, 2011.

[45] V.-T. Pham, M. Bohme, and A. Roychoudhury, "AFLNET: A Greybox Fuzzer for Network Protocols," in *2020 IEEE 13th International Conference on Software Testing, Validation and Verification (ICST)*. Porto, Portugal: IEEE, Oct. 2020, pp. 460–465.

[46] J. Platt, "Probabilistic outputs for support vector machines and comparisons to regularized likelihood methods," *Advances in Large-Margin Classifiers*, vol. 10, no. 3, pp. 61–74, 1999.

[47] J. Rabet, "Adventures in buttplug penetration (testing)," DEF CON 27, Las Vegas, Tech. Rep., 2019.

[48] M. Rodda, A. Chambers, and A. Rohl, "Robust automated event detection from machine-learning analysis of network data," vol. 7, no. 4, pp. 599–599.

[49] R. Rothe, R. Timofte, and L. V. Gool, "DEX: Deep EXpectation of Apparent Age from a Single Image," in *2015 IEEE International Conference on Computer Vision Workshop (ICCVW)*. Santiago, Chile: IEEE, Dec. 2015, pp. 252–257.

[50] I. Rouf, H. Mustafa, M. Xu, W. Xu, R. Miller, and M. Gruteser, "Neighborhood watch: Security and privacy analysis of automatic meter reading systems," in *Proceedings of the 2012 ACM Conference on Computer and Communications Security*. Raleigh North Carolina USA: ACM, Oct. 2012, pp. 462–473.

[51] N. Schiller, M. Chlosta, M. Schloegel, N. Bars, T. Eisenhofer, T. Scharnowski, F. Domke, L. Schönherr, and T. Holz, "Drone Security and the Mysterious Case of DJI's DroneID," in *Proceedings 2023 Network and Distributed System Security Symposium*. San Diego, CA, USA: Internet Society, 2023.

[52] I. Sharafaldin, A. H. Lashkari, S. Hakak, and A. A. Ghorbani, "Developing realistic distributed denial of service (DDoS) attack dataset and taxonomy," in *2019 International Carnahan Conference on Security Technology (ICCST)*, 2019, Conference Proceedings, pp. 1–8.

[53] Q. Shi, J. Shao, Y. Ye, M. Zheng, and X. Zhang, "Lifting Network Protocol Implementation to Precise Format Specification with Security Applications," in *Proceedings of the 2023 ACM SIGSAC Conference on Computer and Communications Security*. Copenhagen Denmark: ACM, Nov. 2023, pp. 1287–1301. [Online]. Available: https://dl.acm.org/doi/10.1145/3576915.3616614

[54] Q. Shi, X. Xu, and X. Zhang, "Extracting protocol format as state machine via controlled static loop analysis."

[55] 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, pp. 1–17, 2018.

[56] SMIA, "Swedish Defence Research Agency Information Warfare Lab," download.netresec.com/pcap/smia-2011, 2011.

[57] M. Smith, D. Moser, M. Strohmeier, V. Lenders, and I. Martinovic, "Economy Class Crypto: Exploring Weak Cipher Usage in Avionic Communications via ACARS," in *Financial Cryptography and Data Security*, A. Kiayias, Ed. Cham: Springer International Publishing, 2017, vol. 10322, pp. 285–301.

[58] M. Spencer, "Pidgin," pidgin.im, 1998.

[59] A. Tridgell, "Samba," www.samba.org, 1992.

[60] Y. Wang, Xiaochun Yun, M. Z. Shafiq, L. Wang, A. X. Liu, Z. Zhang, D. Yao, Y. Zhang, and L. Guo, "A semantics aware approach to automated reverse engineering unknown protocols," in *20th IEEE International Conference on Network Protocols (ICNP)*. Austin, TX, USA: IEEE, Oct. 2012, pp. 1–10.

[61] C. Yang, C. Fu, Y. Qian, Y. Hong, G. Feng, and L. Han, "Deep learning-based reverse method of binary protocol," in *Security and Privacy in Digital Economy*. Singapore: Springer Singapore, 2020, pp. 606–624.

[62] Y. Ye, Z. Zhang, F. Wang, X. Zhang, and D. Xu, "NetPlier: Probabilistic Network Protocol Reverse Engineering from Message Traces," in *Network and Distributed System Security Symposium*. Virtual: Internet Society, 2021.

[63] B. Zadrozny and C. Elkan, "Transforming classifier scores into accurate multiclass probability estimates," in *8th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. Edmonton Alberta Canada: ACM, Jul. 2002, pp. 694–699.

[64] S. Zhao, J. Wang, S. Yang, Y. Zeng, Z. Zhao, H. Zhu, and L. Sun, "ProsegDL: Binary Protocol Format Extraction by Deep Learning-based Field Boundary Identification," in *30th IEEE International Conference on Network Protocols (ICNP)*. Lexington, KY, USA: IEEE, Oct. 2022, pp. 1–12.

[65] F. Zosso and Proto17, "DJI DroneID RF analysis," https://github.com/proto17/dji_droneid/tree/main.

IX. Appendix A

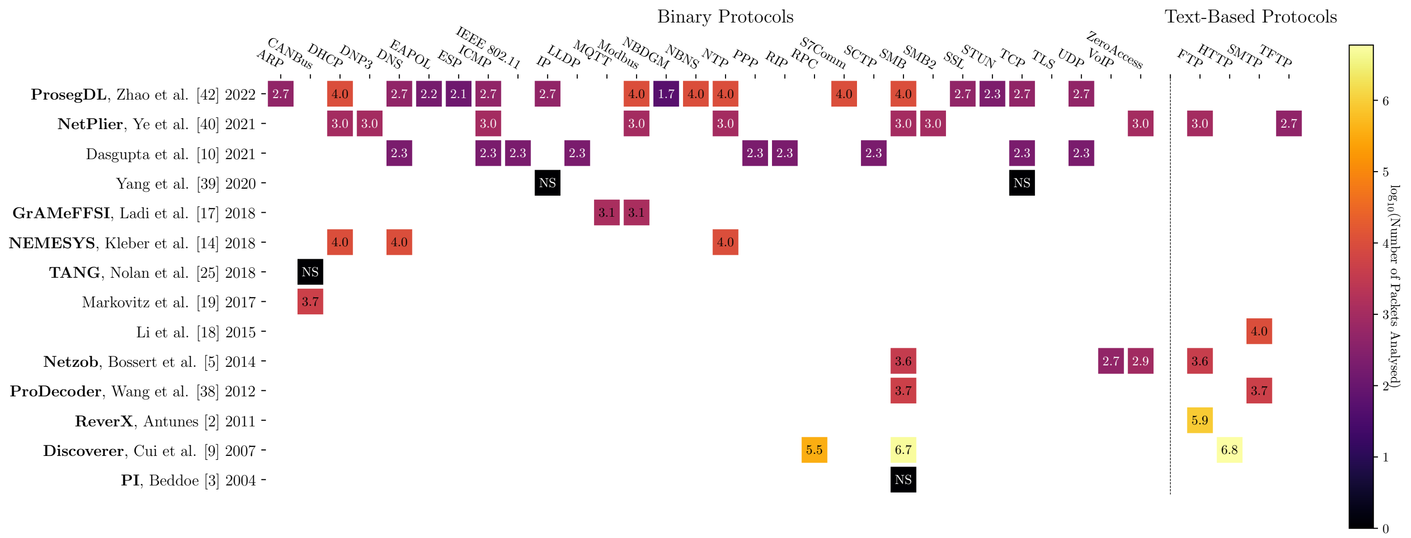


Fig. 11: **Protocols by paper timeline:** for the listed papers on the y-axis, we plot which protocols were evaluated. The value in each cell indicates the $\log_{10}$ number of packets evaluated, where 'NS' indicates this was not specified. Only [62], [64] and [17] evaluated on more than four protocols.