

Toward Unsupervised Protocol Feature Word Extraction

Zhuo Zhang, Zhibin Zhang, Patrick P. C. Lee, Yunjie Liu, and Gaogang Xie

Abstract—Protocol feature words are byte subsequences within traffic payload that can distinguish application protocols, and they form the building blocks of many constructions of deep packet analysis rules in network management, measurement, and security systems. However, how to systematically and efficiently extract protocol feature words from network traffic remains a challenging issue. Existing approaches like those based on n -gram or *Common String* (CS), which simply breaks payload into equal-length pieces or attempts to find a frequent itemset, are ineffective in capturing the hidden statistical structure of the payload content. In this paper, we propose *ProWord*, an unsupervised approach that extracts protocol feature words from traffic traces. *ProWord* builds on two nontrivial algorithms. First, we propose an unsupervised segmentation algorithm based on the modified Voting Experts algorithm, such that we break payload into candidate words according to entropy information and provide more accurate segmentation than existing n -gram and CS approaches. Second, we propose a ranking algorithm that incorporates different types of well-known feature word retrieval heuristics, such that we can build an ordered structure on the candidate words and select the highest ranked ones as protocol feature words. We compare *ProWord* and existing prior approaches via evaluation on real-world traffic traces. We show that *ProWord* captures true protocol feature words more accurately and performs significantly faster.

Keywords—Network traffic analysis, network traffic identification, protocol reverse engineering, unsupervised information extraction.

I. INTRODUCTION

To deal with the increasing variety and complexity of modern Internet traffic, operators often need deep understanding of applications running in their networks. Today's operators are challenged by how to keep pace with the explosive growth of new web and mobile applications [35]. *Protocol feature words* (or feature words for short) are byte subsequences within payload that can distinguish application protocols. If we consider each protocol as a type of communication language, feature words make up a lexicon and form the building blocks for any deep packet analysis. Feature words are important in security and measurement systems. For example, the Linux application classifier L7-Filter [1] uses layer-7 feature words to build

regular expressions for traffic identification. Intrusion detection systems, such as Snort [8] and Bro [3], need feature words to construct rules and guide their engines to properly conduct application layer protocol processing. Traffic analysis tools such as Wireshark [11] and NetDude [5] require third-party development of additional plugins to provide feature support for new protocols. Compared with the noise-prone and easily morphed behavioral features such as packet sizes and interval times, feature words are more stable and distinguishable in traffic classification related applications [13], [16].

However, existing studies on protocol feature word discovery, or in machine learning terms the *feature engineering process*, critically depend on manual labors when protocol specifications are undocumented. When performing protocol reverse engineering, we need many prior experiences to discover feature word boundaries and select candidates as feature words from continuous payload. Text-based protocols, such as SMTP and FTP, contain human-readable feature words, and word boundaries in general can be identified by common delimiters such as whitespaces. However, in the realm of binary protocols, extracting feature words becomes challenging for humans without grammar and syntax prompt. Even worse, we cannot easily tell whether a traffic trace belongs to a text or binary protocol if the protocol is totally unknown. Thus, generating effective rules to identify traffic is labor and experience intensive. For example, L7-Filter pattern files, which include regular expressions built with feature words, are contributed by many researchers and developers worldwide. This motivates us to investigate how to integrate protocol reverse engineering experiences into algorithmic design, so as to automatically extract feature words from network traffic.

A. Related Work and Their Limitations

Traditionally, the intuition behind feature word extraction is based on frequent itemset mining. That is, we believe that feature words must be the substrings that appear more frequently than others. Although Apriori [24] is the most natural choice, it cannot scale well as it needs multiple scans of the original traffic traces. Thus, many other studies have been raised, which usually break continuous payload into small blocks and can be regarded as an attempt to build a bag-of-words model. For text protocols, using whitespaces to delimit feature words [15] is a good choice, but is clearly ineffective for binary protocols.

The n -gram approach can be regarded as a variation of Apriori. It has been widely used to extract feature words in both text and binary protocols [19], [20], [25], [26], [29], [38], [39], such that it uses a sliding window of size n bytes to break payload into equal-length pieces. However, it can tear a feature word larger than n bytes into different pieces, or squish noise

Z. Zhang, Z.B. Zhang, Y. Liu and G. Xie are with the Institute of Computing Technology, Chinese Academy of Sciences, Beijing, P.R. China (e-mail: zhangzhuo@ict.ac.cn, zhangzhibin@ict.ac.cn, liuyj@chinaunicom.cn, xie@ict.ac.cn).

P.P.C. Lee is with the Chinese University of Hong Kong, Hong Kong, China (e-mail: pcleee@cse.cuhk.edu.hk).

Manuscript received December 31, 2013; revised April 1, 2014.

This work was supported in part by the National Basic Research Program of China with Grant 2012CB315801, the NSF of China (NSFC) with Grants 61133015 and 61202411, National Key Technology Support Program 2012BAH01B03, and the Strategic Priority Research Program of CAS with Grant XDA06010303. The work of P.P.C. Lee was supported in part by GRF CUHK 413711 from the Research Grant Council of Hong Kong. (Corresponding author: G. Xie)

A preliminary version of this paper appeared in [44].

bytes into one piece with a shorter one. Recent experimental studies show that n -gram analysis quickly becomes ineffective when capturing relevant content features in moderately varying traffic [22].

Common substring extraction is another popular approach for feature words extraction [34], [36], [37], [40], [42], [43]. Inspired by sequence alignment in bioinformatics for DNA analysis, this approach will find the most common substrings within flows or packets. It selects substrings with a minimum length and a minimum coverage in the trace. Although it can identify words of various lengths with given frequencies, the result may include too many redundancies if we improperly set the minimum length or the minimum coverage. For example, if a substring RCPT TO is a *common substring* that meets the requirement of minimum coverage, its subset items like {RCPT T, RCPT, CPT, RCP, TO, ...} can also be included in our results if we set the minimum length as 2. Some redundancy reduction methods can be applied. The most natural one is the *longest common substring* (LCS) approach, which only selects the longest one from a set of common substrings¹. For {RCPT TO, RCPT, CPT, RCP, TO, ...}, we only choose RCPT TO as the final result. An obvious problem is that, some useful substrings with short length are always excluded if they happen to be a part of another longer one. For example, DATA, EXDATA are two feature words in the SMTP protocol and its extension [6], but DATA will be ignored as EXDATA is the longest common substring. To remove redundancies, Wong *et al.* [41] propose an algorithm for discovering biological non-induced patterns (or substrings) from sequences, and it excludes redundant patterns (or substrings) by statistically induction instead of selecting the longest common ones. However, the single threshold is empirical and limited for redundancy reduction. SANTaClass [36], [37] proposes different rules to filter redundant common terms, but some of the rules, such as removing terms unrelated to applications and removing bad terms, require detailed knowledge of application protocols and hence manual interventions seem inevitable.

To wrap up, the prior studies have two limitations. The first limitation is that they are parameter sensitive approaches. We must select the parameters properly to reach useful results. The parameters, such as n for n -gram or the length and frequency thresholds for CS or LCS, have strong dependencies with final results. However, it is very difficult for an engineer to accurately build the prior knowledge. As a substitute, an engineer with experiences of traffic and protocol analysis may believe how possible that a feature word appears in *ranges*, rather than exact values, of length, frequency, or position within a trace. Thus, the first problem is how to inject this implicit knowledge into real traffic analysis and make the process insensitive with parameters.

The second limitation is that the prior approaches cannot scale well for large-scale traffic traces. For example, the

common substring approaches usually keep the information about substrings, say their frequencies, in a generalized suffix tree [21], which can explode when facing a large volume of data. In network traffic, most substrings appear only once but they can occupy the most memory. Thus, the second problem is how to filter these low frequent items out and save memory for latent useful ones.

Supervised machine learning approaches have been widely used in traffic classification. Most studies focus on designing effective classification algorithms based on state-of-the-art learning tools like support vector machines [18], [28] and Naive Bayesian classifier [12], [33]. Supervised learning approaches require a training set to classify traffic accurately, and they do not give us suggestions on feature generation or selection. In this work, we focus on designing an unsupervised learning approach.

B. Our Contributions

We formulate the protocol reverse engineering problem as an information retrieval problem. We design *ProWord*, a lightweight unsupervised mechanism that automatically and accurately extracts from traffic traces a set of byte subsequences that are most likely to be feature words. *ProWord* addresses two major challenges: (i) how to identify word boundaries within traffic traces to extract candidate feature words and (ii) how to rank byte subsequences such that the ones that are more likely to be feature words will be assigned higher rank scores. To address the first challenge, our idea originates from a segmentation approach in natural language processing, in which texts are divided into meaningful units based on statistical models. As the target network protocol may have unknown specifications, we leverage unsupervised segmentation that discovers word boundaries based on the statistics such as entropy or frequency. Specifically, our work builds on the Voting Experts (VE) algorithm [14], which identifies possible word boundaries using entropy. For example, for the message "MAIL FROM:<a@gmail.com>\r\n" in SMTP payload, our partition result can be the set of "MAIL FROM:<", "a@", "gmail", ".com", and ">\r\n". Compared with existing n -gram approaches, such as the 3-gram partition {MAI, AIL, IL_, L_F, _FR, FRO, ROM, OM:, M:<, ...}, we respect the hidden statistical structure when recognizing word boundaries. Since the baseline VE algorithm can lead to memory explosion, we enhance the VE algorithm with less memory usage by filtering low-frequency subsequences, thereby making the algorithm scalable to high-volume traffic payload.

To address the second challenge, we need to construct an ordered structure on the set of candidate words obtained from our segmentation. Our idea is inspired by the heuristics in information retrieval such as TF-IDF weighting [17], and we adapt such heuristics into traffic analysis. *ProWord* uses a ranking algorithm that maps different dimensions of protocol feature heuristics (e.g., frequency, occurrence location, and length) into different word scoring functions and uses the aggregate score to rank the candidates. To maintain the compactness of our final result, *ProWord* filters any redundant candidates that are very similar to some higher ranked words, and returns the top k candidates as the resulting feature words.

¹The formal definition of the longest common substring (LCS) approach is to find the longest substring that appears in all input strings. The LCS approach in our description can be viewed as a variant of the formal definition, since it first extracts all (common) substrings that meets the minimum coverage requirement and then picks the longest (common) substring.

ProWord is designed as an offline analysis tool that extracts feature words from packet traces. Compared with n -gram approaches, ProWord makes a trade-off of using more memory space to keep track of occurrences of candidate feature words in the VE algorithm, so as to achieve more accurate feature word extraction. Nevertheless, our modified VE algorithm significantly reduces the memory usage for practical use.

In summary, we propose ProWord for unsupervised feature word extraction and make three key contributions.

- *Segmentation on payload*: To our knowledge, this is the first work that adapts a segmentation approach from natural language processing into traffic analysis. We present a novel unsupervised segmentation algorithm that divides payload into a set of candidate words with respect to the hidden statistical structure, while reducing the memory usage for scalable traffic analysis.
- *Ranking on candidate words*: We transform feature word selection into a ranking problem based on our word selection experience and the actual word properties. We propose a ranking algorithm that integrates different dimensions of prior knowledge about feature words. The algorithm also filter any redundancy to maintain the compactness of the returned set of feature words.
- *Evaluation*: We conduct extensive trace-driven evaluation. Using six protocols of different types, we compare ProWord with existing n -gram and common substring approaches. Our results show that ProWord provides more accurate feature word extraction. ProWord also performs significantly faster than the state-of-the-art n -gram approach ProDecoder [38], which requires extensive computations to combine equal-length pieces into meaningful feature words.

The rest of this paper proceeds as follows. In Section II, we describe how we extend the VE algorithm for segmentation. In Section III, we describe our ranking model that incorporates various heuristics. In Section IV, we conduct trace-driven evaluation and compare ProWord with other prior approaches. Finally, Section V concludes the paper.

II. SEGMENTATION

We explore how we generate candidate feature words from a continuous stream of payload. The challenge is how to recognize word boundaries, namely *segmentation*, when no lexicon is available for word recognition.

A. Background: Voting Experts

ProWord builds on the Voting Experts (VE) algorithm [14], which is an unsupervised segmentation approach in natural language processing. It is a local greedy algorithm that operates by sliding a relatively small window along a continuous input stream and selecting the most possible boundary positions for word partitioning. We leverage the VE algorithm to identify the subsequences that are potentially feature words. In this subsection, we first provide an overview of the baseline VE algorithm.

The VE algorithm takes the votes of two experts as inputs. One expert specifies the *word internal entropy* (denoted by

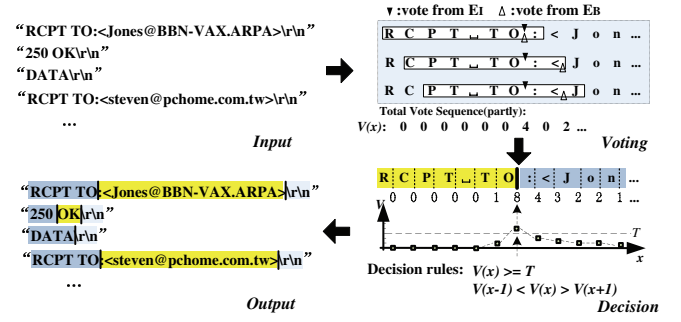


Fig. 1: Overview of the VE algorithm.

H_I), following the intuition that if a subsequence always occurs as a whole in a stream, then it should be retained in entirety. H_I is defined as:

$$H_I(w) = -\log P(w), \quad (1)$$

where $P(w)$ is the occurrence probability of subsequence w within a given stream. A low H_I means that w usually occurs in entirety and has a high probability to be a word.

Another expert specifies the *word boundary entropy* (denoted by H_B), following the intuition that if the successor byte of a subsequence has many variations, then we should put a word boundary in between. H_B is defined as:

$$H_B(w) = -\sum_{c \in \mathcal{C}} P(c|w) \log P(c|w), \quad (2)$$

where $\mathcal{C}$ is the set of all possible successor bytes following a subsequence w , and $P(c|w)$ is the occurrence probability of byte c following w . For example, consider the input sequence "DATA.DAT". The set $\mathcal{C}$ of subsequence "DA" only has one element 'T', while that of "A" has two elements 'T' and '.'. H_B estimates the amount of uncertainty of the bytes after a subsequence. A high H_B indicates that the byte after w varies aggressively and the point after w has a high probability to be a word boundary.

In order to compare these statistical measures among subsequences of different lengths, we normalize them among all subsequences with the same length and denote their normalized values as $E_I(w) = (H_I(w) - \bar{H}_I)/\sigma_I$ and $E_B(w) = (H_B(w) - \bar{H}_B)/\sigma_B$, where $\bar{H}$ and σ denote the mean and standard deviation, respectively.

Figure 1 illustrates the VE algorithm. There are two key phases: *voting* and *decision*. In the voting phase, each expert will vote one position as a possible boundary within each sliding window. The sliding window size, which we denote by L , enables us to generate words of length less than or equal to L . Suppose that i is the offset of the beginning of the sliding window. The internal voting point x_i^I and the boundary voting point x_i^B at offset i can be represented as:

$$x_i^I = \arg \min_{x_i^I = i+j} (E_I(w_{i,i+j}) + E_I(w_{j+1,i+L})), \quad (3)$$

$$x_i^B = \arg \max_{x_i^B = i+j} E_B(w_{i,i+j}), \quad (4)$$

where $j \in (0, L]$, and $w_{a,b}$ represents the subsequence between

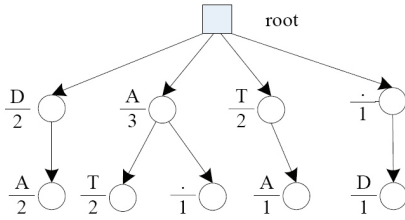


Fig. 2: The 2-depth Trie produced by “DATA.DAT” in the VE algorithm.

offsets a and b inclusively within the input sequence. Each point x has a vote score $V(x)$, which can be computed as:

$$V(x) = \sum_i (\mathbf{1}(x = x_i^I) + \mathbf{1}(x = x_i^B)), \quad (5)$$

where $\mathbf{1}(\cdot)$ is the indicator function such that $\mathbf{1}(x = y) = 1$ if $x = y$; otherwise $\mathbf{1}(x = y) = 0$ if $x \neq y$.

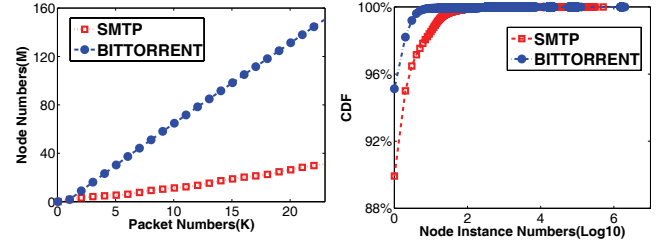
In the decision phase, we identify a point x as a word boundary if the following two rules are met: (i) if the point x obtains more votes than its neighbors (i.e., $V(x) > V(x-1)$ and $V(x) > V(x+1)$) and (ii) if its number of votes exceeds some pre-defined threshold T (i.e., $V(x) > T$).

To illustrate both voting and decision phases, consider the example in Figure 1. Suppose that the input sequence is “RCPT TO:<Jon...”. In the voting phase, the experts E_I and E_B vote a position as the possible boundary. For example, in the first sliding window, both E_I and E_B vote the same position right after “RCPT TO”; in the second and third sliding windows, E_I votes the position after “RCPT TO”, while E_B votes the position after “RCPT TO:<”. If we collect votes for the first three sliding windows, we can get the numbers of votes as “0, 0, 0, 0, 0, 0, 4, 0, 2...”, as shown in Figure 1. In the decision phase, suppose that we obtain the final $V(x)$ as “0, 0, 0, 0, 0, 1, 8, 4, 3, 2, 2, 1...”, after all sliding windows are considered (see Figure 1), and that $T = 6$. Then only the point after “RCPT TO”, which has eight votes, meets both rules. Thus, we put a boundary at that point.

To implement the VE algorithm with sliding window size L , we use an $(L+1)$ -depth Trie (or prefix tree) to hold all possible byte combinations occurring in the stream, in which we can calculate the entropy values accordingly. We set the Trie depth as $L+1$ since we need one more byte to calculate the word boundary entropy H_B for the longest possible word of length L . Each node at level i of the Trie corresponds to a subsequence of length i . The children of a node have a common prefix in their ascendants. Figure 2 shows how the sequence “DATA.DAT” produces a 2-depth Trie. The number at each node records the occurrence count of the subsequence. For example, for all the three occurrences of ‘A’, there are two possible successors in a window of size two, namely ‘.’ (with two occurrences) and ‘T’ (with one occurrence).

B. Limitations of the Baseline VE Algorithm

Although the VE algorithm has been successfully used in natural language processing, it is not a scalable approach in traffic analysis. In the Trie constructed in the baseline VE



(a) Node space size versus number of processed packets (b) CDF of node frequency

Fig. 3: Node frequency statistics in Trie construction.

algorithm, the number of descendants of each node is at most its alphabet size. In natural language processing, the actual bound in reality is far less than the theoretical one because the conventional language combinations can limit the occurrence of some subsequences. For example, in English, when we meet “tio” at the end of a word, we will predict the next character to be ‘n’ with a very high probability as “-tion” is a common combination.

On the other hand, when analyzing traffic payload, especially for pure binary data, the kind of conventional language combination is very unlikely to occur. In other words, the probability mass of subsequences can be distributed more *sparsely* in network traffic. A more sparse data stream tends to produce more new byte combinations, meaning that the stream is more difficult to compress and needs more memory to store and manipulate. In particular, this sparsity problem will lead to the node space explosion in our algorithm design.

To illustrate the node space explosion, we conduct some experimental studies on the frequency distributions of the Trie nodes when analyzing the SMTP and BITTORRENT traffic traces with an 8-byte window Trie (see the trace descriptions in Section IV). The key observation is that some protocols may have very high sparsity. As shown in Figure 3(a), as a binary protocol, BITTORRENT has significantly more Trie nodes than SMTP and is much more sparse. Specifically, for 20K packets from real traffic, BITTORRENT itself produces 130 million nodes, which account for over 5GB of memory.

Nevertheless, a majority of nodes have very low frequency counts. Figure 3(b) plots the frequency distribution of all Trie nodes. It shows that 95% of nodes only occur once or twice throughout the entire trace. Thus, although capturing all nodes requires a huge space, we only need to focus on a small subset of them that have sufficiently high frequencies. We use this as a guideline to address the node space explosion problem.

C. Pruning And Compensation for the VE algorithm

To address the node space explosion problem when applying the VE algorithm to traffic analysis, we propose a *pruning* step for Trie construction so as to limit the memory usage. The intuition is that subsequence nodes with very low frequencies are unlikely to be the true feature words. Thus, we can prune the node space by periodically removing nodes with frequencies below some pre-defined threshold.

Our pruning step builds on the Lossy Counting Algorithm [31], an approximate counting algorithm that can return frequency estimations of high-frequency data items over a data stream using limited memory by periodically removing low-frequency items. In this step, one key parameter is the maximum probability estimation error ϵ , which limits our estimation error within $[0, \epsilon]$. Intuitively, we only prune a node if its corresponding subsequence has an occurrence probability no more than ϵ . Since any new byte introduces $(L + 1)$ new subsequence instances, we can use ϵ to compute the pruning period $M = \lceil 1/(\epsilon \times (L + 1)) \rceil$ and the pruning frequency threshold $\theta = i \times M \times (L + 1) \times \epsilon \approx i$, where i is the number of periods that have been processed. As each newly added subsequence in the i^{th} period may be removed before, its estimated frequency in this period has a difference with the true one. Thus, we add an error $(i - 1)$ to each Trie node as in [31], which is the maximum possible error to the frequency. We will remove a subsequence if its frequency plus error is no more than $\theta \approx i$ times in the current i^{th} period.

There is a trade-off between the estimation error and memory cost. A low ϵ (or a high M) can retain more subsequence nodes, but will increase the risk of running out of memory in a period, while a high ϵ may lose useful frequency information. In practice, since we do not determine the actual subsequence distribution for a given trace, we can set M as large as possible, as long as it does not induce the out-of-memory error in a period. Then we can determine ϵ and θ accordingly.

Our pruning step removes the low-frequency subsequences from the Trie, but we cannot exclude these subsequences for certain when computing the entropy values in the VE algorithm. In other words, we need to consider every subsequence within the sliding window in the entropy computations, regardless of if the subsequence appears in the Trie or has been pruned. Here, we propose a *compensation* step. Here, we assign an occurrence probability $\epsilon/2$ to each subsequence that does not appear in the Trie, since the occurrence probability of any pruned subsequence is at most ϵ due to the maximum estimation error.

Algorithm 1 shows the entire segmentation algorithm, which generates a set of candidate feature words for a given protocol trace $\mathcal{P}$. The function BUILDTRIE computes the occurrence frequencies for the subsequences over each pruning period (lines 7-17) and deletes nodes if their corresponding subsequences have frequencies below the pruning frequency threshold i (lines 18-22). The segmentation algorithm first constructs a Trie and computes the entropy values for all subsequences in the Trie (lines 29-30). For each packet, it slides a window over the packet payload. Within a sliding window, the algorithm first runs the compensation step on the pruned subsequences (line 34) and computes the votes (line 35). It finally identifies the boundaries (lines 37-42) and extracts all candidate feature words into $\mathcal{W}$ (line 43).

We analyze the computational complexity of the segmentation algorithm, which dominates the load of ProWord (see Section IV-D). Its complexity is mainly due to the Trie construction, which has two key operations: frequency counting and periodic pruning. Consider an $(L + 1)$ -depth Trie for m bytes of input payload. Frequency counting takes place at

Algorithm 1 Segmentation Algorithm

```

1: Input: Protocol trace  $\mathcal{P}$ ; Pruning period  $M$ ; Sliding window length  $L$ ; Decision threshold  $T$ 
2: Output: Candidate word set  $\mathcal{W}$ 
3: TrieNode ( $w, f, e$ ):  $w$  = subsequence,  $f$  = estimated frequency of  $w$ , and  $e$  = maximum error to  $f$ 
4: function BUILDTRIE( $\mathcal{P}, M, L$ )
5:   Trie  $\mathcal{T} = \emptyset$ 
6:   for  $i^{th}$  period of  $M$  bytes  $\mathcal{P}_i$  for pruning do
7:     for each subsequence  $w$  with length  $\leq (L + 1)$  in  $\mathcal{P}_i$  do
8:        $f_i \leftarrow$  frequency count of  $w$  in  $\mathcal{P}_i$ 
9:        $v \leftarrow$  vertex of  $w$  in  $\mathcal{T}$ 
10:      if  $v = \text{none}$  then
11:         $e \leftarrow i - 1$ 
12:         $v \leftarrow (w, f_i, e)$ 
13:        Insert  $v$  to  $\mathcal{T}$ 
14:      else
15:         $v.f \leftarrow v.f + f_i$ 
16:      end if
17:    end for
18:    for all  $v$  in  $\mathcal{T}$  do
19:      if  $v.f + v.e \leq i$  then
20:        Delete  $v$  from  $\mathcal{T}$ 
21:      end if
22:    end for
23:  end for
24:  return  $\mathcal{T}$ 
25: end function
26:
27: procedure SEGMENTATION( $\mathcal{P}, M, L, T$ )
28:    $\epsilon = 1/(M \cdot (L + 1))$ 
29:    $\mathcal{T} = \text{BUILDTRIE}(\mathcal{P}, M, L)$ 
30:   Compute  $E_I, E_B$  for all subsequences in  $\mathcal{T}$ 
31:    $\mathcal{W} = \emptyset$ 
32:   for all packet in  $\mathcal{P}$  do
33:     while sliding window of  $L$  through the packet do
34:       Compute  $E_I, E_B$  for pruned subsequences with occurrence probability  $\epsilon/2$ 
35:       Compute votes according to  $E_I, E_B$ 
36:     end while
37:     Compute votes into  $V$ 
38:     for all vote point  $x$  do
39:       if  $V(x) > V(x - 1)$  and  $V(x) > V(x + 1)$  and  $V(x) > T$  then
40:         Set a boundary at  $x$ 
41:       end if
42:     end for
43:     Insert all words between boundaries to  $\mathcal{W}$ 
44:   end for
45: end procedure

```

most $m(L + 1)$ times since any new byte can introduce at most $(L + 1)$ subsequence instances into counting. Periodic pruning traverses all nodes in the Trie for pruning at each pruning period. The total number of pruning operations can be upper bounded by $7m(L + 1)$ [31]. To sum up, the total complexity of Trie construction is $O(mL)$, which is actually also the complexity of ProWord.

III. RANKING

We may extract millions of candidate feature words from the VE algorithm. It is important to identify the words of interest that can be used as protocol features in traffic analysis. Our goal is to rank the extracted candidate words and select

the top k of them that are most likely to be feature words. Specifically, we construct an ordered structure on the candidate words with a score function that aggregates various attribute values of individual words based on our prior knowledge on protocol features. As a proof of concept, here we exploit three attributes that are widely used in real-life protocol reverse engineering, namely *frequency*, *location*, and *length*. First, a feature word is expected to occur in most packets or flows. To distinguish it from others, we identify the candidate words that have relatively high frequencies. Second, we identify the candidate words that appear at a relatively fixed location from the beginning or the end of a packet or flow, since these words may be used as part of the protocol definitions. Finally, a possible feature word should have a length within a reasonable range, since a very short word has weak distinguishability while a very long word costs much resource in communication. Thus, we define an aggregate score function (denoted by F_{agg}) of word w by combining all above heuristics as follows:

$$F_{agg}(w) = F_{freq}(w) \cdot F_{loc}(w) \cdot F_{len}(w), \quad (6)$$

where $F_{freq}(w)$, $F_{loc}(w)$, and $F_{len}(w)$ denote the score functions for the attributes frequency, location, and length, respectively. In this way, we can rank all candidate words by comparing their aggregate scores. Here, we combine the individual score functions by multiplication, as it represents proportional fairness [27] among the score functions and brings the best result based on our experience.

We emphasize that our goal is *not* to find a feature word that scores high in all attributes. For example, the header fields of HTTP may appear in different orders, and hence they may score low in the location score. However, if they still score high in both frequency and location, and hence the aggregate score, then they can still be extracted as feature words, as shown in our evaluation (see Section IV).

A. Score Rules and Score Functions

To construct the score functions, we formally define intuitive and reasonable score rules that should be satisfied when determining the protocol features. In this work, our score rules and score functions build on the information retrieval heuristics proposed for ranking web pages [17]. The novelty of our work is to adapt the heuristics into the context of traffic analysis. In particular, when we adapt the heuristics, we must respect the specific properties of network protocols in general, so as to accurately extract the feature words from traffic traces.

Rules for the frequency score function. Let $\mathcal{W}$ be the candidate word set. For $w \in \mathcal{W}$, let $X_t(w)$ be the total number of occurrences of w in all packets, and $X_p(w)$ be the number of packets containing w . We define two rules for the frequency score function as follows.

Rule 1: For $w_1, w_2 \in \mathcal{W}$, suppose that $X_t(w_1) = X_t(w_2)$. If $X_p(w_1) > X_p(w_2)$, then $F_{freq}(w_1) > F_{freq}(w_2)$. $\square$

Rule 2: For $w_1, w_2 \in \mathcal{W}$, suppose that $X_p(w_1) = X_p(w_2)$. If $X_t(w_1) > X_t(w_2)$, then $\frac{X_t(w_1)}{X_t(w_2)} F_{freq}(w_2) > F_{freq}(w_1) > F_{freq}(w_2)$. $\square$

These two rules use $X_t(w)$ and $X_p(w)$ as two inputs for $F_{freq}(w)$. We would like to select a word occurring in most

packets or flows. In other words, we are interested in finding how many packets or flows can be covered if we take a word as a feature word. Here, we only discuss the packet coverage of a word (i.e., number of packets containing the word), while the idea can be easily extended to flow coverage (i.e., number of flows containing the word). Rule 1 states that if two words have the same total number of occurrences, the one with higher packet coverage is more likely to be a feature word; Rule 2 states that if two words have the same packet coverage, we give a higher score to the one with more occurrences. In particular, we expect that F_{freq} follows a sub-linear growth with total number of occurrences of a candidate word if its packet coverage is fixed, since subsequences occurring multiple times within one packet tend to be trivial ones such as padded bytes and we should limit the score growth due to a high number of occurrences. Here, we define $\frac{X_t(w_1)}{X_t(w_2)}$ as the linear factor that bounds the growth of the score function. For example, there is a packet segmented as “AB|AB|AB|AB|AB|CD”, where subsequences “AB” and “CD” appear five times and once, respectively. Then, the frequency score of a word in “AB” should be less than five times that of “CD”. Here, we choose the logarithmic function to define a monotonic and sub-linear function, as the logarithmic function is the most common choice for defining ranking functions in information retrieval [32]. Based on the above two rules, we define $F_{freq}(w)$ as follows:

$$F_{freq}(w) = X_p(w) \cdot (1 + \log \frac{X_t(w)}{X_p(w)}). \quad (7)$$

Rules for the location score function. For a given candidate word set $\mathcal{W}$ and $w \in \mathcal{W}$, let $X_p(w)$ be the number of packets containing w . Also, let $X_m(w)$ be the maximum number of occurrences of w at a given position in all packets (i.e., we count the occurrences of w in each possible position and compute the maximum).

Rule 3: For $w_1, w_2 \in \mathcal{W}$, suppose that $X_p(w_1) = X_p(w_2)$. If $X_m(w_1) > X_m(w_2)$, then $F_{loc}(w_1) > F_{loc}(w_2)$. $\square$

Rule 4: For $w_1, w_2 \in \mathcal{W}$, suppose that $\frac{X_m(w_1)}{X_p(w_1)} = \frac{X_m(w_2)}{X_p(w_2)}$. If $X_p(w_1) > X_p(w_2)$, then $F_{loc}(w_1) > F_{loc}(w_2)$. $\square$

Rules 3 and 4 stem from the intuition on location centrality of feature words, in which we give a high score to a word that appears in relatively fixed locations. Rule 3 captures the basic location centrality heuristic, in which we score higher a word that has more instances on some fixed locations; Rule 4 scores higher a word with more occurrences if two words have same possibilities of occurring at some fixed points, as it shows more observable evidences in the data. Similar to above, we here use a logarithmic function to limit the score growth to be sub-linear and define $F_{loc}(w)$ as follows:

$$F_{loc}(w) = \frac{X_m(w)}{X_p(w)} \cdot \log X_p(w). \quad (8)$$

Rule for the length score function. For a given candidate word set $\mathcal{W}$ and $w \in \mathcal{W}$, let $|w|$ be the length of w (in number of bytes). Let the range $[\delta_l, \delta_h]$ be the preferable length space of feature words. Intuitively, if w is a feature word, its length $|w|$ is likely in the range $[\delta_l, \delta_h]$.

Rule 5: For $w_1, w_2 \in \mathcal{W}$, if $|w_1| \in [\delta_l, \delta_h]$ and $|w_2| \notin$

$[\delta_l, \delta_h]$, then $F_{len}(w_1) > F_{len}(w_2)$. $\square$

Rule 5 presents our heuristic of identifying feature words based on their lengths. We exclude the words that are too short or too long, and hence define a piecewise function as follows:

$$F_{len}(w) = \begin{cases} \frac{|w|}{\delta_l} & \text{if } |w| < \delta_l, \\ 1 & \text{if } \delta_l \leq |w| \leq \delta_h, \\ \frac{\delta_h}{|w|} & \text{if } |w| > \delta_h. \end{cases} \quad (9)$$

The range $[\delta_l, \delta_h]$ can be defined according to prior knowledge. In this work, we set the range as $[2, 10]$.

B. Compactness

Based on our definition of F_{agg} , we can rank the set of candidate words and select the top k ranked words as feature words. Here, k is a very small integer compared to the number of candidate words, and can be chosen by users in real deployment.

On the other hand, when a protocol uses feature words to define semantics, there are some option fields or variations that will induce *redundancies*, which refer to the words that have similar patterns or even the same semantics. These redundancies may show up in the returned top k feature words. For example, “RCPT TO:” and “RCPT TO” are two common feature words in SMTP indicating a recipient, and due to their minor variations we may add them in our returned results as two different words. Thus, one key requirement is to filter these redundancies and maintain the *compactness* of our resulting feature words.

To compact our results, a straightforward approach is to recognize similar words based on the *edit distance*, defined as the minimum number of edits needed to transform one string into the other, due to the insertion, deletion, or substitution of a single character. Although this metric can reflect the similarity between two words, it can introduce errors to our redundancy filtering. Since protocol feature words are typically some short strings, two words with a small edit distance may actually refer to semantically different words. For example, “250” and “220” have the same edit distance as “RCPT TO:” and “RCPT TO”. However, “250” and “220” are actually different words in SMTP, where “250” is an “okay” reply for a requested mail action, while “220” is a “ready” reply for the mail transfer service.

Hence, we need a conservative strategy for redundancy filtering. In this paper, we use two strict criteria to identify redundancies. First, as a substitute of the edit distance, we check if a word is a substring of another one. Second, as a criterion to distinguish protocol features from common data, we check if the two words begin at the same location within packet payload. Algorithm 2 outlines our ranking algorithm on how we select the top k feature words from a given candidate word set $\mathcal{W}$. The function ISREDUNDANT checks if two words are redundant (lines 3-10). The algorithm first computes the aggregate scores of all words in $\mathcal{W}$ (lines 13-15) and sorts all words in descending order of the aggregate scores (line 16). It then extracts the highest scored words and removes those that are redundant (lines 17-24). Finally, it returns the set of k feature words $\mathcal{F}$.

Algorithm 2 Ranking Algorithm

```

1: Input: Candidate word set  $\mathcal{W}$ ; Number of output words  $k$ 
2: Output: A set of  $k$  feature words  $\mathcal{F}$ 
3: function ISREDUNDANT( $\hat{w}, \mathcal{F}$ )
4:   for all  $w \in \mathcal{F}$  do
5:     if  $\hat{w}$  is a substring of  $w$  and  $\hat{w}$  and  $w$  begin at the same
       location then
6:       return true
7:     end if
8:   end for
9:   return false
10: end function
11:
12: procedure RANKING( $\mathcal{W}, k$ )
13:   for all  $w \in \mathcal{W}$  do
14:      $F_{agg}(w) \leftarrow F_{freq}(w) \cdot F_{loc}(w) \cdot F_{len}(w)$ 
15:   end for
16:   Sort  $\mathcal{W}$  in descending order of  $F_{agg}(w)$ 
17:    $\mathcal{F} = \emptyset$ 
18:   while  $\mathcal{F}$  has less than  $k$  elements and  $\mathcal{W} \neq \emptyset$  do
19:      $\hat{w} \leftarrow$  the highest scored word in  $\mathcal{W}$ 
20:     if ISREDUNDANT( $\hat{w}, \mathcal{F}$ ) = false then
21:        $\mathcal{F} \leftarrow \mathcal{F} + \{\hat{w}\}$ 
22:     end if
23:      $\mathcal{W} \leftarrow \mathcal{W} - \{\hat{w}\}$ 
24:   end while
25: end procedure

```

TABLE I: Summary of network protocols used in evaluation.

Protocol	Size(B)	Packet	Flow
SMTP	81,366K	95,068	547
POP3	92,077K	101,253	719
FTP	7,032K	71,068	4,549
HTTP	65,423K	48,601	1,386
BITTORRENT	50,169K	62,613	1,260
TONGHUASHUN	3,020K	9,453	165

IV. EVALUATION

We evaluate ProWord on several widely used level-7 protocols. We classify the protocols into two groups. The first group has publicly available official specifications, which we use as ground truths to identify the true feature words. The second group has no specifications that document feature words, but there exist effective rules for our verification. For example, L7-Filter contains hundreds of rules that were manually built by volunteers and can serve as references for our validation.

We collect traffic traces from a university network gateway. We select six protocols shown in Table I. The protocols SMTP, POP3, FTP, and HTTP have their specifications available in the online RFC documents. They are all text-based protocols. BITTORRENT [2] is a peer-to-peer file sharing protocol whose official specifications are available but different client applications often have their own variations in implementation. TONGHUASHUN [9] is one of the most popular stock trade applications in China. It was recorded with over one hundred million users in early 2012. While its payload is encrypted, it has identifiable patterns at the head of its flows. Both BITTORRENT and TONGHUASHUN are binary protocols.

ProWord has a few tunable parameters as shown in Table II.

TABLE II: Values of tunable parameters.

Parameter	Value
Window size in byte L in VE	10
Vote threshold T in VE	6
Processing bytes M in a pruning period	10,000,000
Preferable word length $[\delta_l, \delta_h]$ in byte	[2,10]

We point out that the window size and the vote threshold, both of which are used in the VE algorithm, are related to different language properties but not sensitive to the result. Here, we set their values based on our experience.

A. Evaluation on Protocol Feature Word Extraction

We compare ProWord with state-of-the-art approaches. Firstly, we compare ProWord with existing n -gram approaches on feature word extraction [25], [26], [29], [38], [39]. We consider three ranking approaches based on n -gram partition: (1) frequency statistics test (e.g., in [26], [29], [38]), which selects the words that have the highest frequencies of occurrences; (2) two-sample Kolmogorov-Smirnov (K-S) test (e.g., in [25], [39]), which selects the words that have the most similar distributions on different traces; and (3) ProDecoder [38], a recently proposed approach that attempts to capture the latent dependencies of n -grams and performs the selection with the help of topic modeling. To choose n for n -gram, we note that a larger n can generate sparse frequency distributions [38]. Thus, we choose $n = 3$ in our evaluation. In addition, we also compare ProWord with the approach (denoted by “VE+Freq”) that uses the VE algorithm for unsupervised word segmentation (see Section II) but uses the frequency statistics test to rank candidate words. This enables us to evaluate the effectiveness of ProWord in combining different types of heuristics to rank different feature words.

Secondly, we also compare ProWord with typical common substring (CS) extraction approaches [34], [36], [37], [40], [42], [43]: (1) the baseline CS approach, which selects all substrings with a minimum length and a minimum coverage in the trace; (2) the longest common substring (LCS) approach, which only selects the longest one among the set of results extracted with CS; and (3) $LCS + 64B$, in which we limit LCS to focus on the first 64 bytes to each packet or flow (we focus on packets in this paper and it can be extended to flows easily). We choose $LCS + 64B$ for two reasons. First, prior studies [23], [30], [43] conclude that this approach is a competitive choice for feature words analysis. Second, as LCS is usually implemented with the generalized suffix tree, which implies higher space complexity for deeper inspection, using the first 64 Bytes of payload is a natural trade-off choice. Furthermore, we choose the minimum coverage as 1% and set the minimum length with 2, as this setting gives the best result in parameter selection based on our evaluation.

To fairly compare the accuracy of ProWord and state-of-the-art approaches, we are interested in two metrics:

- *Number of true feature words*: We measure the number of true feature words in a set of top k candidates, where k is the input parameter. The ranked results are considered to

be effective if the number is high. As n -gram approaches cannot output a whole word as long as n is less than the original word length, we check their results manually and score a hit if all pieces of a true feature word appear in the top k list.

- *Conciseness* [15]: We measure the ratio of the number of polymorphic candidates to the number of true feature words. Two words are polymorphic to each other if either they are identical or one of them is a substring of another. For example, in the top 10 candidates, if the true feature words are {RCPT TO, MAIL FROM} while there are three polymorphic candidates {RCPT TO, RCPT TO:, MAIL FROM}, then the conciseness value is calculated as $3/2 = 1.5$. The conciseness metric captures how frequently a true feature word and its variants appear in the final results of top k candidates. It is desirable to have a lower conciseness value, meaning that our results have fewer redundancies.

Figure 4 shows the results for the four protocols SMTP, POP3, FTP, and HTTP, whose official specifications provide ground truths of feature words. For the number of true feature words (see Figures 4(a)-(d)), the VE-based approaches (i.e., “VE+Freq” and ProWord) are more effective than the n -gram ones since the former ones identify word boundaries more accurately while the latter ones always divide words into equal-length pieces. ProWord returns more feature words than “VE+Freq” since it includes more selection criteria in addition to frequency. The y -axis of Figures 4(a)-(d) also shows the actual number of feature word that appear in our traces, and we find that ProWord can detect 82-94% of feature words, significantly higher than other approaches. For conciseness (see Figures 4(e)-(h)), VE-based approaches also have lower conciseness than n -gram ones, and ProWord further reduces the conciseness of “VE+Freq” by 12%.

The number of true feature words that can be captured heavily depends on the available traces and the number of feature words in protocol specification in addition to the value of k . We point out that although ProWord only identifies around 13-18 true feature words in the top 100 list, these feature words can actually cover the protocol trace with a very high accuracy. For each protocol we consider, 98% of packets contains at least one of the feature words identified by ProWord. Furthermore, we dig into the non-feature words returned. For HTTP, we find that a majority of them are format marks (e.g., “\r\n”, “:/”), conventional words (e.g., “google”, “com”), and random strings (e.g., padding bytes or numbers). We can filter them easily through manual inspection.

Figure 5 shows the number and conciseness of true feature words identified comparison between ProWord and various common substring approaches. For the number of true feature words (see Figures 5(a)-(d)), ProWord outperforms the CS-based approaches (i.e., CS, LCS, and $LCS + 64B$) by 2-3 times. There are two reasons to explain this result. First, CS-based approaches will rank higher many noise words that are induced by some more trivial substrings. Take LCS for example, \r\n is a very trivial substring in text-based traffic, while substrings induced by it like n\r\n or s\r\n (which usually appear at

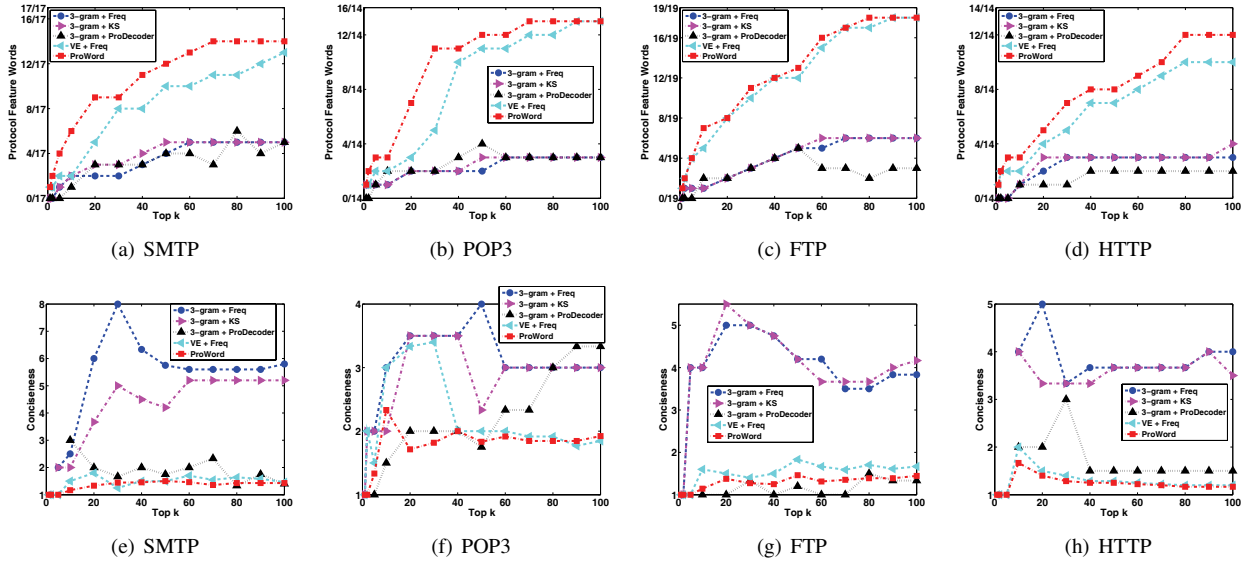


Fig. 4: Number of true feature words captured (figures (a)-(d)) and conciseness (figures (e)-(h)) versus k for SMTP, POP3, FTP, and HTTP. For the y -axis of figures (a)-(d), we also show the actual number of feature words that appear in our traces.

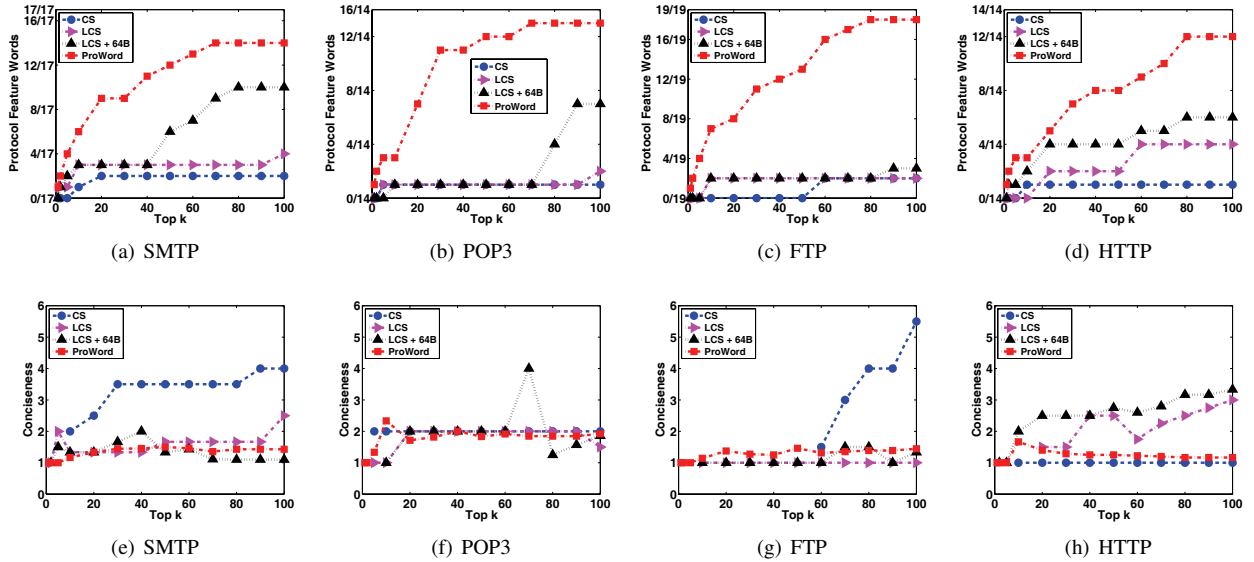


Fig. 5: Comparison between ProWord and approaches based on common substrings. Number of true feature words captured (figures (a)-(d)) and conciseness (figures (e)-(h)) versus k for SMTP, POP3, FTP, and HTTP. For the y -axis of figures (a)-(d), we also show the actual number of feature words that appear in our traces.

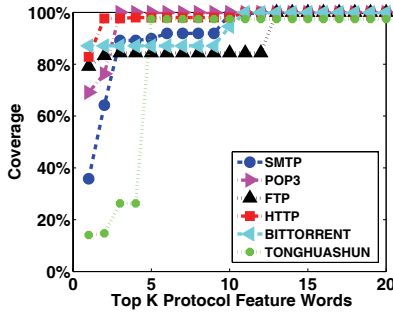
the end of a line and n or s are the ending letters of many words) can also be assigned a higher rank in *LCS*. Thus, *LCS* will assign high ranks to many redundant words. However with ProWord, it may include $\backslash r \backslash n$ only once and its induced substrings can be filtered out during payload segmented in VE. Second, ProWord adopts an effective ranking mechanism, which comprehensively take frequency, location, and length of a candidate word into account. For conciseness (see Figures 5(e)-(h)), although ProWord has a close conciseness with CS-based approaches in the low top k s, it is more stable with the increase of the number of extracted feature words.

In addition to official specifications, L7-Filter rules

also provide some ground truths. Table III shows the rank comparisons for capturing and ranking a specific set of feature words we consider based on L7-Filter rules. We also consider the binary protocols BITTORRENT and TONGHUASHUN, whose feature words we choose are “dl:ad2:id20:” and “\xfd\xfd\xfd\xfd\x30\x30\x30\x30\x30” respectively. Here rank 1 refers to the highest. A smaller rank value implies that a word is more likely to be excluded from the top- k list for small k . We see that ProWord gives a higher rank than other approaches, especially for long feature words of BITTORRENT and TONGHUASHUN. In addition, ProWord further reduces the rank range of “VE+Freq” by

TABLE III: Ranks for different feature words based on L7-Filter rules.

Protocol ("feature words")	3-gram +Freq	3-gram +KS	3-gram +ProDecoder	CS	LCS	LCS +64B	VE +Freq	ProWord
SMTP("220")	37	34	25±5	160	223	51	12	4
POP3("OK")	1	1	4±1	1	6	7	1	1
POP3("-ERR")	114	113	>100	>500	>500	>500	43	7
FTP("FTP")	162	160	>100	440	203	233	47	25
HTTP("HTTP")	63	60	>100	69	16	1	3	2
BITTORRENT	>500	>500	>100	>500	149	17	8	4
TONGHUASHUN	>500	>500	>100	462	55	55	8	6

Fig. 6: Flow coverage of top k protocol feature words in ProWord.

about 36% on average.

B. Evaluation on Flow Coverage of Protocol Feature Words in ProWord

In the previous subsection, we argue that ProWord has high packet coverage. Here, we show that ProWord also has high flow coverage, defined as the percentage of flows that contain one keyword identified by ProWord. Figure 6 shows the flow coverage for all protocols we consider (including text and binary protocols) versus the number of top candidates being selected. We see that if we set $k = 15$ (i.e., the top 15 candidates), ProWord can cover almost all flows.

C. Evaluation on Ranking Model

To evaluate the effect of the ranking functions used in ProWord, we compare different ranking functions and their combinations using the HTTP trace. Figure 7 shows the results. All feature words (x -axis) get higher ranks (i.e., smaller rank values) with the frequency score function used in ProWord compared to the results obtained from the pure frequency function (PureF) that simply counts the occurrences, as ProWord frequency function will score higher to a word occurring in most packets or flows. Also, when combining all three score functions as ProWord, all feature words rank even higher and are more easily distinguished.

D. Evaluation on Running Speed

We evaluate the running speed of ProWord. We benchmark ProWord on a server that has four Intel Xeon CPUs running at 2.50GHz with 16GB RAM. Table IV summarizes the running speeds (in KB/s) of both segmentation and ranking phases.

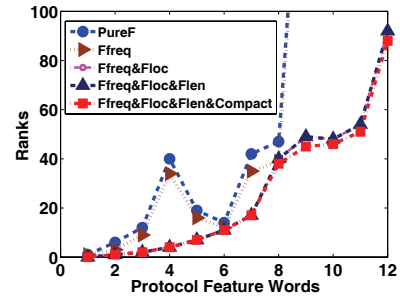


Fig. 7: Comparison of different combinations of score functions in the ranking model.

TABLE IV: Running speeds (in KB/s) of ProWord.

Protocol	Segmentation	Ranking
SMTP	13.8	1,255
POP3	16.0	1,787
FTP	16.7	2,344
HTTP	11.8	2,128
BITTORRENT	16.5	2,144
TONGHUASHUN	12.7	1,399

We see that segmentation has a significantly lower speed than ranking, and dominates the overall load of ProWord. The running speed of segmentation is 10-20KB/s. Note that ProWord is designed as an offline analysis tool and its running speed is lower than the network line rate.

Nevertheless, ProWord runs significantly faster than the state-of-the-art n -gram approach ProDecoder [38], and hence allows more scalable analysis. ProDecoder is evaluated on a testbed with similar hardware configurations to as ours, and it needs almost 3 hours for keywords inference of 5,000 SMTP packets with a total of 340KB (see Table I of [38]). This translates to a running speed of only 0.31KB/s, while ProWord achieves 13.8KB/s, which is at least 40 times faster. The main reason is that ProDecoder, which builds on n -gram, breaks feature words into pieces. It needs more computational cycles to recover the correlation among the pieces and rebuild the feature words. On the other hand, ProWord uses a more lightweight approach for segmentation.

E. Evaluation on Space Usage

Recall from Section II-C that ProWord uses the Lossy Counting Algorithm (LCA) [31] to prune the Trie so as to limit the memory requirement while minimizing the errors of

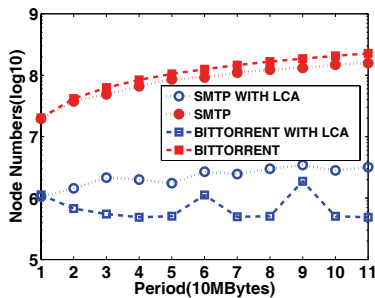


Fig. 8: Node space reduction with Lossy Counting Algorithm (LCA).

frequency estimation. Here, we evaluate the memory saving of ProWord when using LCA. Figure 8 shows the results for the protocols SMTP and BITTORRENT. We see that LCA reduces the number of Trie nodes by an order of magnitude. Also, LCA maintains the number at a low level even after the traces have been processed for a long time.

One tradeoff that ProWord makes is to require more memory space than n -gram approaches. In comparison, for the BITTORRENT trace, n -gram approaches only need about 200MB of memory, while ProWord uses 3GB after pruning the Trie. On the other hand, n -gram approaches cost significantly more time than ProWord to get useful results.

F. Evaluation on Hybrid Traffic

To the best of our knowledge, all prior protocol feature extraction approaches have a strong assumption that they require a single-protocol trace as the input. That is, we must pre-process the trace data so that only the packets or flows that belong to the target protocol are retained and all unrelated packets or flows are filtered. In the following, we use real traffic trace composed of a mix of different protocols as the input and show how ProWord can play in a complicated environment.

We collect a new trace from our institute gateway. The trace lasts for one hour. It contains 32GB of traffic composed of 43M packets and 875K flows. To provide a ground truth for the trace, we first apply protocol classification to it using conventional rules like transport layer ports and L7-filter rules. Figure 9 shows the 5-tuple flow-level composition of the trace. We see that the top 3 protocols include DNS, BitTorrent, and HTTP. The Link Local Multicast Name Resolution (LLMNR) [4] is a domain name resolution protocol based on DNS for both IPv4 and IPv6 hosts on the same local link. The Simple Service Discovery Protocol (SSDP) [7] is a network protocol for advertisement and discovery of network services and presence information. Corel_VNC [10] is a protocol for graphical desktop sharing provided by Canadian developer Corel. To our knowledge, the trace has a larger volume and is more diverse than those being used in the evaluation of prior protocol feature extraction approaches.

In our evaluation, we do not conduct any preprocessing, but instead directly run ProWord on the trace and examine the robustness of ProWord. Table V presents the top 20 feature

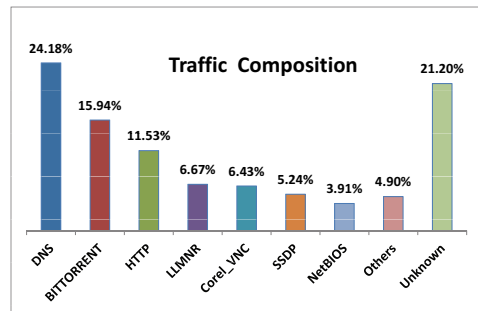


Fig. 9: Protocols composition in hybrid traffic trace.

words output by ProWord, as well as their flow coverage (i.e., percentage of all flows in the trace that contain the corresponding feature word) and main protocol source. We find that the top 20 feature words cover about 78% of all flows. Note that the encrypted protocol *Corel_VNC* has no feature words found in ProWord. Similar to the previous results, the running speed is around 15.9KB/s and the space usage is 2.8GB. Therefore, we conclude that ProWord still works as expected even in a hybrid trace with a mix of different protocols.

V. CONCLUSIONS

This paper presents ProWord, an unsupervised approach that automatically extracts protocol feature words from network traffic traces. It builds on a modified word segmentation algorithm to generate candidate feature words, while limiting the memory space by filtering low-frequency subsequences. It also builds on a ranking algorithm that incorporates protocol reverse engineering experiences into extracting the top-ranked feature words, and removes redundancies to maintain the compactness of the results. Trace-driven evaluation shows that ProWord is more effective than n -gram and common substring approaches, in terms of accuracy and speed, in extracting feature words from real-life protocol traces. Our work explores a design space of how the domain knowledge of natural language processing can be adapted into traffic analysis.

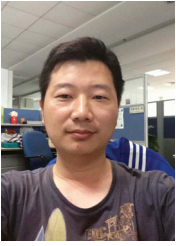
REFERENCES

- [1] "Application layer packet classifier for linux." [Online]. Available: <http://l7-filter.sourceforge.net/>
- [2] "Bittorrent - delivering the world's content." [Online]. Available: <http://www.bittorrent.com/>
- [3] "The bro network security monitor." [Online]. Available: <http://bro-ids.org>
- [4] "Link local multicast name resolution protocol." [Online]. Available: <http://en.wikipedia.org/wiki/LLMNR>
- [5] "The network dump data displayer and editor." [Online]. Available: <http://netdude.sourceforge.net>
- [6] "Simple mail transfer protocol rfc 5321." [Online]. Available: <http://datatracker.ietf.org/doc/rfc5321/>
- [7] "Simple service discovery protocol." [Online]. Available: <http://en.wikipedia.org/wiki/SSDP>
- [8] "Snort network intrusion detection system." [Online]. Available: <http://www.snort.org>
- [9] "Tong hua shun financial services network." [Online]. Available: <http://www.10jqka.com.cn/>

TABLE V: Top 20 feature words extracted from hybrid traffic trace

Ranks	Keywords	Flow Coverage(*100%)	Main Protocol Source
1	"d1:ad2:id20:"	12.74%	BITTORRENT
2	"\x03com\x00\x00\x01\x00\x01"	11.38%	DNS
3	"6:target20:"	10.26%	BITTORRENT
4	"GET /"	10.10%	HTTP
5	"HTTP/1.1"	12.11%	HTTP
6	"\x04wpad\x00\x00\x01\x00\x01"	5.34%	LLMNR
7	"\x1c\x00\x01"	9.99%	DNS&LLMNR
8	"Host: "	4.34%	HTTP&Others
9	"\xc0\x0c\x00\x05\x00\x01\x00\x00"	8.50%	BITTORRENT
10	"\x00\x00\x01\x00\x01"	17.30%	DNS&NetBIOS
11	"\x02cn\x00\x00\x01\x00\x01"	3.82%	DNS
12	"239.255.255.250:1900/r/n"	2.79%	SSDP
13	"d1:rd2:id20:"	2.79%	BITTORRENT
14	"\x04wpad\x00\x00\x01"	4.06%	LLMNR
15	"\x05baidu"	3.64%	DNS
16	"M-SEARCH *"	2.44%	SSDP
17	"\x02\x00\x00\x00\x00"	3.67%	DNS
18	"9:info_hash20:"	1.91%	BITTORRENT
19	"2:ip4:\x9f\xe2+"	1.88%	BITTORRENT
20	"POST /"	1.87%	HTTP

- [10] "Video network computing." [Online]. Available: <http://en.wikipedia.org/wiki/VNC>
- [11] "Wireshark network protocol analyzer." [Online]. Available: <http://www.wireshark.org>
- [12] D. Bonfiglio, M. Mellia, M. Meo, D. Rossi, and P. Tofanelli, "Revealing skype traffic: when randomness plays with you," in *Proc. of ACM SIGCOMM*, 2007.
- [13] A. Callado, C. Kamienski, G. Szabó, B. Gero, J. Kelner, S. Fernandes, and D. Sadok, "A survey on internet traffic identification," *IEEE Communications Surveys & Tutorials*, vol. 11, no. 3, pp. 37–52, 2009.
- [14] P. Cohen and N. Adams, "An algorithm for segmenting categorical time series into meaningful episodes," in *Advances in Intelligent Data Analysis*. Springer, 2001, pp. 198–207.
- [15] W. Cui, J. Kannan, and H. J. Wang, "Discoverer: Automatic protocol reverse engineering from network traces," in *Proc. of USENIX Security*, 2007.
- [16] A. Dainotti, A. Pescapè, and K. C. Claffy, "Issues and future directions in traffic classification," *IEEE Network*, vol. 26, no. 1, pp. 35–40, 2012.
- [17] H. Fang, T. Tao, and C. Zhai, "A formal study of information retrieval heuristics," in *Proc. of ACM SIGIR*, 2004.
- [18] A. Finamore, M. Mellia, M. Meo, and D. Rossi, "Kiss: Stochastic packet inspection classifier for udp traffic," *IEEE/ACM Trans. on Networking*, vol. 18, no. 5, pp. 1505–1515, 2010.
- [19] G. Gu, P. Porras, V. Yegneswaran, M. Fong, and W. Lee, "Bothunter: Detecting malware infection through ids-driven dialog correlation," in *Proc. of USENIX Security*, 2007.
- [20] G. Gu, J. Zhang, and W. Lee, "Botsniffer: Detecting botnet command and control channels in network traffic," in *Proc. of NDSS*, 2008.
- [21] D. Gusfield, *Algorithms on strings, trees and sequences: computer science and computational biology*. Cambridge University Press, 1997.
- [22] D. Hadžiosmanović, L. Simionato, D. Bolzoni, E. Zamboni, and S. Etalle, "N-gram against the machine: on the feasibility of the n-gram network analysis for binary protocols," in *Proc. of Research in Attacks, Intrusions, and Defenses (RAID)*, 2012.
- [23] P. Haffner, S. Sen, O. Spatscheck, and D. Wang, "ACAS: automated construction of application signatures," in *Proc. of ACM MineNets*, 2005.
- [24] J. Han, M. Kamber, and J. Pei, *Data mining: concepts and techniques*. Morgan kaufmann, 2006.
- [25] N. Hentehzadeh, A. Mehta, V. K. Gurbani, L. Gupta, T. K. Ho, and G. Wilathgamuwa, "Statistical analysis of self-similar session initiation protocol (sip) messages for anomaly detection," in *IFIP Int. Conf. on New Technologies, Mobility and Security (NTMS)*, 2011.
- [26] A. Jamdagni, Z. Tan, X. He, P. Nanda, and R. P. Liu, "Repids: A multi tier real-time payload-based intrusion detection system," *Computer Networks*, vol. 57, pp. 811–824, 2013.
- [27] F. Kelly, A. Maulloo, and D. Tan, "Rate control in communication networks: shadow prices, proportional fairness and stability," in *Journal of the Operational Research Society*, vol. 49, 1998.
- [28] Z. Li, R. Yuan, and X. Guan, "Accurate classification of the internet traffic based on the svm method," in *Proc. of IEEE ICC*, 2007.
- [29] W. Lu, G. Rammedi, and A. A. Ghorbani, "Clustering botnet communication traffic based on n-gram feature selection," *Computer Communications*, vol. 34, no. 3, pp. 502–514, 2011.
- [30] J. Ma, K. Levchenko, C. Kreibich, S. Savage, and G. M. Voelker, "Unexpected means of protocol inference," in *Proc. of ACM IMC*, 2006.
- [31] G. S. Manku and R. Motwani, "Approximate frequency counts over data streams," in *Proc. of VLDB*, 2002.
- [32] C. D. Manning, P. Raghavan, and H. Schütze, *Introduction to information retrieval*. Cambridge university press Cambridge, 2008, vol. 1.
- [33] A. W. Moore and D. Zuev, "Internet traffic classification using bayesian analysis techniques," in *Proc. of ACM SIGMETRICS*, 2005.
- [34] B.-C. Park, Y. J. Won, M.-S. Kim, and J. W. Hong, "Towards automated application signature generation for traffic identification," in *Proc. of IEEE/IFIP NOMS*, 2008.
- [35] A. Tongaonkar, R. Keralapura, and A. Nucci, "Challenges in network application identification," in *Proc. of USENIX LEET*, 2012.
- [36] —, "SANTaClass: A self adaptive network traffic classification system," in *Proc. of IFIP Networking*, 2013.
- [37] A. Tongaonkar, R. Torres, M. Iliofotou, R. Keralapura, and A. Nucci, "Towards self adaptive network traffic classification," *Computer Communications*, 2014.
- [38] 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 *Proc. of IEEE ICNP*, 2012.
- [39] Y. Wang, Z. Zhang, D. D. Yao, B. Qu, and L. Guo, "Inferring protocol state machine from network traces: a probabilistic approach," in *Proc. of ACNS*, 2011.
- [40] Y. Wang, Y. Xiang, W. Zhou, and S. Yu, "Generating regular expression signatures for network traffic classification in trusted network management," *Journal of Network and Computer Applications*, vol. 35, no. 3, pp. 992–1000, 2012.
- [41] A. K. Wong, D. Zhuang, G. C. Li, and E.-S. A. Lee, "Discovery of non-induced patterns from sequences," in *Pattern Recognition in Bioinformatics*, 2010.
- [42] M. Ye, K. Xu, J. Wu, and H. Po, "Autosig-automatically generating signatures for applications," in *Proc. of IEEE Computer and Information Technology (CIT)*, 2009.
- [43] J. Zhang, Y. Xiang, W. Zhou, and Y. Wang, "Unsupervised traffic classification using flow statistical properties and ip packet payload," *Journal of Computer and System Sciences*, 2012.
- [44] Z. Zhang, Z. Zhang, P. P. C. Lee, Y. Liu, and G. Xie, "ProWord: An unsupervised approach to protocol feature word extraction," in *Proc. of IEEE INFOCOM*, 2014, <http://www.cse.cuhk.edu.hk/~pcclee/www/pubs/infocom14proword.pdf>.



Zhuo Zhang is a Ph.D. candidate at the Institute of Computing Technology, Chinese Academy of Sciences (ICT/CAS). His research interests include network traffic classification and protocol reverse engineering.



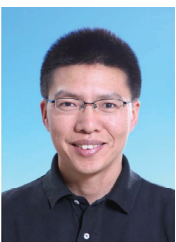
Zhibin Zhang is currently an associate professor at the Institute of Computing Technology, Chinese Academy of Sciences (ICT/CAS). His research interests include network security, traffic classification, distributed system, data mining and machine learning.



Patrick P. C. Lee is an assistant professor of the Department of Computer Science and Engineering at the Chinese University of Hong Kong. He currently heads the Advanced Networking and System Research Laboratory and is working very closely with a group of graduate students on different projects in networks and systems. His research interests are in various applied/systems topics including cloud computing and storage, distributed systems and networks, operating systems, and security/resilience.



Yunjie Liu is a member of the China Academy of Engineering, director of the Science and Technology Committee of China Unicom, and dean of School of Information and Communication Engineering at Beijing University of Posts and Telecommunications. His main areas of research are Internet architecture and system. Currently, He is also the chief scientist of networking in ICT/CAS.



Gaogang Xie is a professor and director of Network Technology Research Center at Institute of Computing Technology, Chinese Academy of Sciences (ICT/CAS). His research interests include future Internet architecture, software-defined networking, Internet measurement.