ProWord: An Unsupervised Approach to Protocol Feature Word Extraction

Zhuo Zhang[1,2], Zhibin Zhang[1], Patrick P. C. Lee[3], Yunjie Liu[4], and Gaogang Xie[1]
[1]Institute of Computing Technology, Chinese Academy of Sciences (CAS), China
[2]University of CAS, China, [3]The Chinese University of Hong Kong, Hong Kong, China
[4]Beijing University of Posts and Telecommunications, China
{zhangzhuo,zhangzhibin,xie}@ict.ac.cn, pclee@cse.cuhk.edu.hk, liuyj@chinaunicom.cn

***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 n-gram approaches simply break payload into equal-length pieces and 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 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 n-gram approaches via evaluation on real-world traffic traces. We show that ProWord captures true protocol feature words more accurately and performs significantly faster.**

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 [1]. *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 [2] uses layer-7 feature words to build regular expressions for traffic identification. Intrusion detection systems, such as Snort [3] and Bro [4], need feature words to construct rules and guide their engines to properly conduct application layer protocol processing. Traffic analysis tools such as Wireshark [5] and NetDude [6] require third-party development of additional plugins to provide feature support for new protocols. Compared to 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 [7], [8].

978-1-4799-3360-0/14/$31.00 © 2014 IEEE

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

Existing studies on feature word extraction usually break continuous payload into small units to build a bag-of-words model. Using whitespaces to delimit feature words [9] is clearly ineffective for binary protocols. Instead, n-gram has been widely used to extract feature words in binary protocols [10]–[16], 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 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 [17]. Thus, our primary objective is to address the limitations of existing n-gram approaches and provide more accurate feature word extraction.

Supervised machine learning has 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], [19] and Naive Bayesian classifier [20], [21]. Supervised learning approaches require a training set to classify traffic accurately, and 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 [22], 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 to 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 [23], 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.

ProWord is designed as an offline analysis tool that extracts feature words from packet traces. Compared to 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 filters 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 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 [14], 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 [22], 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 H_I), following the intuition that if a subsequence always occurs as a whole in a stream, it should be retained in entirety. H_I is defined as:

$$H_I(w) = -\log P(w), \tag{1}$$

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:

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

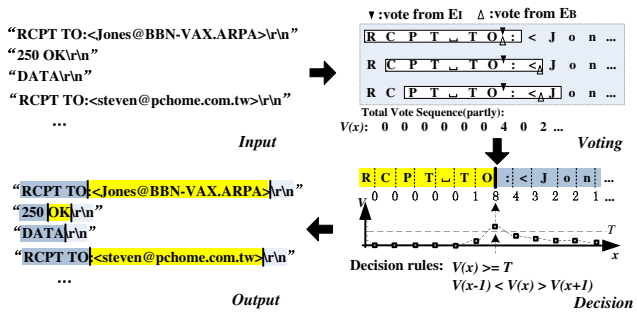


Fig. 1. Overview of the VE algorithm.

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 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 $\mathbf{1}(.)$ is the indicator function such that $\mathbf{1}(x = y) = 1$ if $x = y$, or 0 otherwise.

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 the input sequence is "RCPT TO:<Jon...". In the voting phase, if we collect votes for the first three sliding windows, we can get "$0, 0, 0, 0, 0, 0, 4, 0, 2 \cdots$"; finally, we obtain the final $V(x)$ as "$0, 0, 0, 0, 0, 1, 8, 4, 3, 2, 2, 1 \cdots$". In the decision phase, suppose $T = 6$. Only the point after "RCPT TO" with eight votes meets both rules, and we put a boundary at that point.

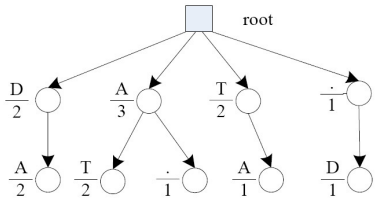


Fig. 2. The 2-depth Trie produced by "DATA.DAT" in the VE algorithm.

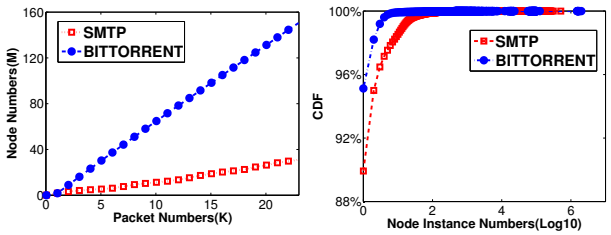
To implement the VE algorithm with sliding window size L, we use an $(L+1)$-depth Trie (or prefix tree) to hold all possible byte combinations occurring in the stream, in which we can calculate the entropy values accordingly. We set the Trie depth as $L+1$ since we need one more byte to calculate the 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 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



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

Fig. 3. Node frequency statistics in Trie construction.

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 [24], 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 [24], 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-C). 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 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 $7m(L+1)$ [24]. 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

Algorithm 1 Segmentation Algorithm

Input: Protocol trace $\mathcal{P}$; Pruning period M; Sliding window length L; Decision threshold T
Output: Candidate word set $\mathcal{W}$
TrieNode (w, f, e): w = subsequence, f = estimated frequency of w, and e = maximum error to f
function BUILDTRIE($\mathcal{P}$, M, L)
  Trie $\mathcal{T} = \emptyset$
  for i^{th} period of M bytes $\mathcal{P}_i$ for pruning **do**
    for each subsequence w with length $\leq (L+1)$ in $\mathcal{P}_i$ **do**
      $f_i \leftarrow$ frequency count of w in $\mathcal{P}_i$
      $v \leftarrow$ vertex of w in $\mathcal{T}$
      if v = **none then**
        $e \leftarrow i - 1$
        $v \leftarrow (w, f_i, e)$
        Insert v to $\mathcal{T}$
      else
        $v.f \leftarrow v.f + f_i$
      end if
    end for
    for all v in $\mathcal{T}$ **do**
      if $v.f + v.e \leq i$ **then**
        Delete v from $\mathcal{T}$
      end if
    end for
  end for
  return $\mathcal{T}$
end function

procedure SEGMENTATION($\mathcal{P}$, M, L, T)
  $\epsilon = 1/(M \cdot (L+1))$
  $\mathcal{T} =$ BUILDTRIE($\mathcal{P}, M, L$)
  Compute E_I, E_B for all subsequences in $\mathcal{T}$
  $\mathcal{W} = \emptyset$
  for all packet in $\mathcal{P}$ **do**
    while sliding window of L through the packet **do**
      Compute E_I, E_B for pruned subsequences with occurrence probability $\epsilon/2$
      Compute votes according to E_I, E_B
    end while
    Compute votes into V
    for all vote point x **do**
      if $V(x) > V(x-1)$ **and** $V(x) > V(x+1)$ **and** $V(x) > T$ **then**
        Set a boundary at x
      end if
    end for
    Insert all words between boundaries to $\mathcal{W}$
  end for
end procedure

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:

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

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 [25] 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 [23]. 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 $F_{freq}(w_1) > F_{freq}(w_2)$. □

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 $\frac{X_t(w_1)}{X_t(w_2)} F_{freq}(w_2) > F_{freq}(w_1) > F_{freq}(w_2)$. □

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". 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 $F_{loc}(w_1) > F_{loc}(w_2)$. □

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 $F_{loc}(w_1) > F_{loc}(w_2)$. □

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 $F_{len}(w_1) > F_{len}(w_2)$. □

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 $\mathcal{F}$.

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 [26] 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 [27] is a widely used online stock analysis application. While its payload is encrypted, it has identifiable patterns at the head of its flows. Both BITTORRENT and TONGHUASHUN are binary protocols.

Algorithm 2 Ranking Algorithm

Input: Candidate word set $\mathcal{W}$; Number of output words k
Output: A set of k feature words $\mathcal{F}$
function IsRedundant($\hat{w}, \mathcal{F}$)
  for all $w \in \mathcal{F}$ **do**
    if $\hat{w}$ is a substring of w **and** $\hat{w}$ and w begin at the same location **then**
      return true
    end if
  end for
  return false
end function

procedure Ranking($\mathcal{W}$, k)
  for all $w \in \mathcal{W}$ **do**
    $F_{agg}(w) \leftarrow F_{freq}(w) \cdot F_{loc}(w) \cdot F_{len}(w)$
  end for
  Sort $\mathcal{W}$ in descending order of $F_{agg}(w)$
  $\mathcal{F} = \emptyset$
  while $\mathcal{F}$ has less than k elements **and** $\mathcal{W} \neq \emptyset$ **do**
    $\hat{w} \leftarrow$ the highest scored word in $\mathcal{W}$
    if IsRedundant($\hat{w}, \mathcal{F}$) = **false then**
      $\mathcal{F} \leftarrow \mathcal{F} + \{\hat{w}\}$
    end if
    $\mathcal{W} \leftarrow \mathcal{W} - \{\hat{w}\}$
  end while
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

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]

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

In this subsection, we compare ProWord with existing n-gram approaches on feature word extraction [10]–[14]. We compare ProWord with two ranking approaches based on n-gram partition: frequency statistics test (e.g., in [10], [11], [14]), which selects the words that have the highest frequencies of occurrences, and two-sample Kolmogorov-Smirnov (K-S) test (e.g., in [12], [13]), which selects the words that have the most similar distributions on different traces. To choose n for n-gram, we note that a larger n can generate sparse frequency distributions [14]. 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.

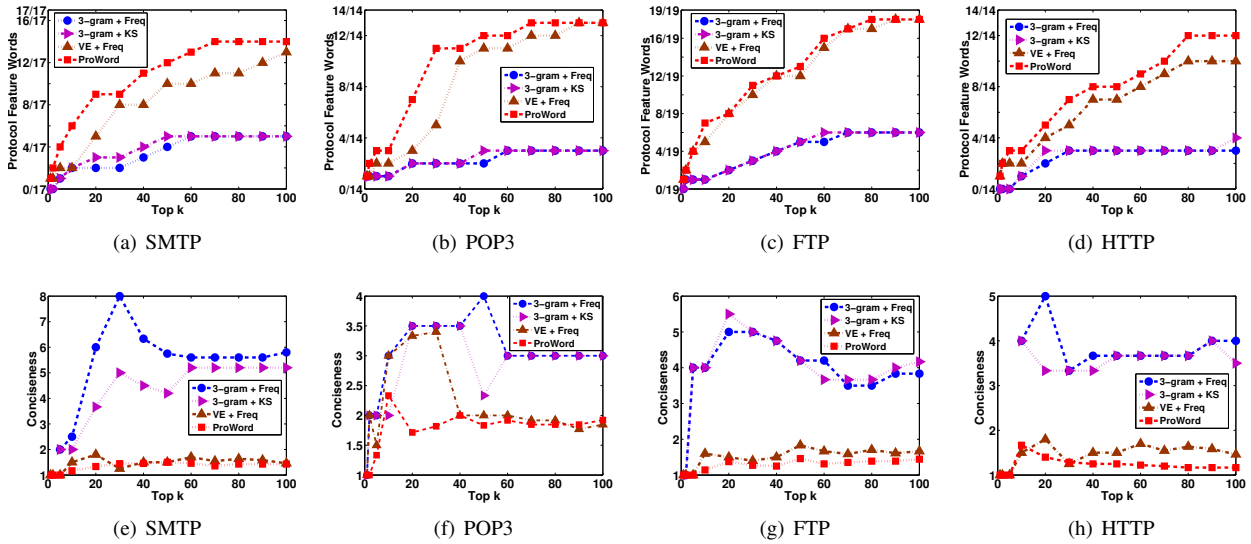
We are interested in two metrics:

- *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* [9]: We measure the ratio of the number of polymorphic candidates to the number of true feature words. This 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 words 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 contain 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.

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 "`d1:ad2:id20:`"



(a) SMTP (b) POP3 (c) FTP (d) HTTP

(e) SMTP (f) POP3 (g) FTP (h) HTTP

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

TABLE III
RANKS FOR DIFFERENT FEATURE WORDS BASED ON L7-FILTER RULES.

Protocol ("feature words")	3-gram +Freq	3-gram +KS	VE +Freq	ProWord
SMTP("220")	37	34	12	4
POP3("+OK")	1	1	1	1
POP3("-ERR")	114	113	43	7
FTP("FTP")	162	160	47	25
HTTP("HTTP")	63	60	3	2
BITTORRENT	>200	>200	8	4
TONGHUASHUN	>200	>200	8	6

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 about 36% on average.

B. 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 5 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.

C. 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

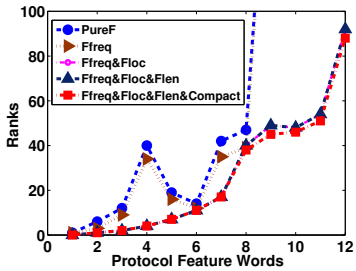


Fig. 5. Comparison of different combinations of score functions in the ranking model.

speeds (in KB/s) of both segmentation and ranking phases. 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 [14], 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 [14]). 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.

D. Evaluation on Space Usage

Recall from Section II-C that ProWord uses the Lossy Counting Algorithm (LCA) [24] to prune the Trie so as to limit the memory requirement while minimizing the errors of frequency estimation. Here, we evaluate the memory saving

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

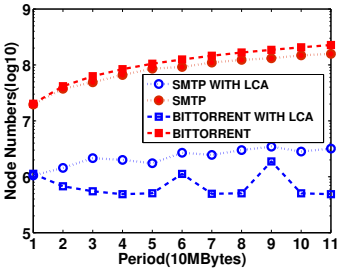


Fig. 6. Node space reduction with Lossy Counting Algorithm (LCA).

of ProWord when using LCA. Figure 6 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.

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 improves the accuracy and speed of existing n-gram approaches in extracting feature words. Our work explores a design space of how the domain knowledge of natural language processing can be adapted into traffic analysis.

ACKNOWLEDGMENTS

This work was supported by the National Basic Research Program of China with Grant 2012CB315801, the NSF of China (NSFC) with Grants 61133015 and 61100171, the National High-tech R&D Program of China with Grant 2013AA013501, Strategic Priority Research Program of CAS under Grant XDA06010303, the National Science & Technology Pillar Program No.2012BAH01B03, and GRF CUHK 413711 from the Research Grant Council of Hong Kong.

REFERENCES

[1] A. Tongaonkar, R. Keralapura, and A. Nucci, "Challenges in network application identification," in *Proc. of USENIX LEET*, 2012.
[2] "Application layer packet classifier for linux." [Online]. Available: http://l7-filter.sourceforge.net/
[3] "Snort network intrusion detection system." [Online]. Available: http://www.snort.org
[4] "The bro network security monitor." [Online]. Available: http://bro-ids.org
[5] "Wireshark network protocol analyzer." [Online]. Available: http://www.wireshark.org
[6] "The network dump data displayer and editor." [Online]. Available: http://netdude.sourceforge.net
[7] 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.
[8] 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.
[9] W. Cui, J. Kannan, and H. J. Wang, "Discoverer: Automatic protocol reverse engineering from network traces," in *Proc. of USENIX Security*, 2007.
[10] W. Lu, G. Rammidi, and A. A. Ghorbani, "Clustering botnet communication traffic based on n-gram feature selection," *Computer Communications*, vol. 34, no. 3, pp. 502–514, 2011.
[11] 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.
[12] 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.
[13] 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.
[14] 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.
[15] G. Gu, J. Zhang, and W. Lee, "Botsniffer: Detecting botnet command and control channels in network traffic," in *Proc. of NDSS*, 2008.
[16] 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.
[17] D. Hadžiosmanović, L. Simionato, D. Bolzoni, E. Zambon, and S. Etalle, "N-gram against the machine: on the feasibility of the n-gram network analysis for binary protocols," in *Research in Attacks, Intrusions, and Defenses*. Springer, 2012, pp. 354–373.
[18] Z. Li, R. Yuan, and X. Guan, "Accurate classification of the internet traffic based on the svm method," in *Proc. of IEEE ICC*, 2007.
[19] 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.
[20] 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.
[21] A. W. Moore and D. Zuev, "Internet traffic classification using bayesian analysis techniques," in *Proc. of ACM SIGMETRICS*, 2005.
[22] 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.
[23] H. Fang, T. Tao, and C. Zhai, "A formal study of information retrieval heuristics," in *Proc. of ACM SIGIR*, 2004.
[24] G. S. Manku and R. Motwani, "Approximate frequency counts over data streams," in *Proc. of VLDB*, 2002.
[25] 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.
[26] "Bittorrent - delivering the world's content." [Online]. Available: http://www.bittorrent.com/
[27] "Tong hua shun financial services network." [Online]. Available: http://www.10jqka.com.cn/