

Received May 10, 2019, accepted June 14, 2019, date of publication June 26, 2019, date of current version July 19, 2019.

Digital Object Identifier 10.1109/ACCESS.2019.2924928

P-Gram: Positional N-Gram for the Clustering of Machine-Generated Messages

JIAOJIAO JIANG¹, STEVE VERSTEEG², JUN HAN²,
MD ARAFAT HOSSAIN², JEAN-GUY SCHNEIDER³, CHRISTOPHER LECKIE⁴,
AND ZEINAB FARAHMANDPOUR²

¹Department of Computer Science and Software Engineering, School of Science, RMIT University, Melbourne, VIC 3000, Australia

²School of Software and Electrical Engineering, Swinburne University of Technology, Melbourne, VIC 3122, Australia

³School of Information Technology, Deakin University, Burwood, VIC 3125, Australia

⁴Department of Computing and Information Systems, The University of Melbourne, Melbourne, VIC 3010, Australia

Corresponding author: Jiaojiao Jiang (chairly2010@gmail.com)

This work was supported by the Australian Research Council Linkage Project LP150100892 “Generating Virtual Deployment Environments for Enterprise Software Systems.”

ABSTRACT An IT system generates messages for other systems or users to consume, through direct interaction or as system logs. Automatically identifying the types of these machine-generated messages has many applications, such as intrusion detection and system behavior discovery. Among various heuristic methods for automatically identifying message types, the clustering methods based on keyword extraction have been quite effective. However, these methods still suffer from keyword misidentification problems, i.e., some keyword occurrences are wrongly identified as payload and some strings in the payload are wrongly identified as keyword occurrences, leading to the misidentification of the message types. In this paper, we propose a new machine language processing (MLP) approach, called *P*-gram, specifically designed for identifying keywords in, and subsequently clustering, machine-generated messages. First, we introduce a novel concept and technique, positional *n*-gram, for message keywords extraction. By associating the position as meta-data with each *n*-gram, we can more accurately discern which *n*-grams are keywords of a message and which *n*-grams are parts of the payload information. Then, the positional keywords are used as features to cluster the messages, and an entropy-based positional weighting method is devised to measure the importance or weight of the positional keywords to each message. Finally, a general centroid clustering method, *K*-Medoids, is used to leverage the importance of the keywords and cluster messages into groups reflecting their types. We evaluate our method on a range of machine-generated (text and binary) messages from the real-world systems and show that our method achieves higher accuracy than the current state-of-the-art tools.

INDEX TERMS Machine-generated messages, positional *n*-gram, clustering.

I. INTRODUCTION

Machine-generated messages are automatically generated by a computer process, application, or other mechanism without the active intervention of a human. For example, IT systems generate highly structured messages and store them in log files. These log files are based on scripts and are crucial for system security audits. IT systems also generate and exchange formatted messages with each other according to certain communication protocols. These messages follow some defined formats, i.e., specific sequences of fixed keywords interleaved with dynamic data fields (the “payload”). One application system or communication protocol often

involves multiple types of messages. The problem of clustering those machine-generated messages is to segregate the messages into structurally similar message clusters, which correspond to *message types* with different formats. In data analytics applications, such as log analysis, network communications analysis and system response emulation, the underlying formats of messages are often not available *a priori*. Separating messages according to their types is a critical step to any further analysis.

Keyword extraction methods using Frequent Pattern Mining (FPM) [6], [8] and Natural Language Processing (NLP) [23], [25] have been developed to cluster machine-generated messages. Examples found in recent literature include Discoverer [8], ReverX [2], AutoReEngine [15], ProDecoder [22], [27], and the HsMM-based method

The associate editor coordinating the review of this manuscript and approving it for publication was Waldemar W. Koczkodaj.

proposed by Cai et al. [7]. These methods have been proven very useful, but still have major limitations in keyword identification and thus message clustering: (1) some keywords appear multiple times in messages, but they are considered only once by existing methods; (2) some payload fields that contain the same string as a certain message keyword are wrongly treated as keyword occurrences; (3) some keywords are missed due to their being part of other longer keywords. These issues are particularly pronounced in *binary* messages where critical keywords can be very short. For example, the message type keywords in the Lightweight Directory Access Protocol (LDAP) have only one Byte [18].

In this paper, we propose a new Machine Language Processing (MLP) approach, called *P-gram*, specifically designed for the clustering of machine-generated messages. P-gram leverages the *positions* of keywords in the template structure of machine-generated messages to extract message keywords more accurately. It is based on the observation that message keywords appear at relatively *fixed* positions in machine-generated messages.

The proposed P-gram consists of four main steps. Given a set of machine-generated messages, in *Step 1*, we introduce our new concept, *positional n-gram*, and identify frequent positional *n*-grams with different *n* lengths in messages. In *Step 2*, we extract the longest common positional *n*-grams and the shorter but significant positional *n*-grams as position-specific keywords (or *positional keywords*) from the frequent positional *n*-grams of the previous step. To reduce the impact of variable-length payloads on keyword position, in *Step 3*, we analyze the variation of the positions of each keyword, calculate the position window(s) for each keyword, and merge the positional keywords within their windows. In *Step 4*, we use the merged keywords as features for message clustering. To numerically weigh each feature in messages, we exploit the position information of keywords and derive a “positional weighting” for each feature using a variability weighting technique, namely, *entropy analysis*. The features’ positional weightings are then used by a general centroid clustering method, *K-Medoids*, to group the messages into clusters reflecting message types.

P-gram effectively addresses the aforementioned *keyword mis-identification* issues faced by existing methods. First, P-gram can identify the *repetitive occurrences* of keywords in messages by analyzing the probability density of keywords at different positions (or position windows). Second, it can effectively filter out from the candidate keywords *noise* that is part of message payload. Third, by using the proposed *independent frequency*, P-gram can effectively distinguish short keywords from those longer ones that contain the short keywords as sub-strings, whereas existing methods cannot because they solely rely on the generic frequency of strings.

In summary, we make the following major contributions:

- By introducing P-gram we effectively address the *keyword mis-identification* issues commonly afflicting existing methods in the clustering of machine-generated messages.

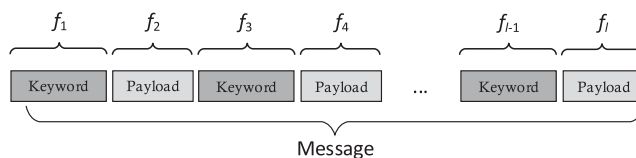


FIGURE 1. An example of a machine-generated message.

- *Theorems* are provided to prove the effectiveness of the proposed P-gram in message keywords extraction.
- P-gram is based on the statistical characterization of machine-generated messages, and it assumes no knowledge of the underlying IT system. Accordingly, it can be applied to both *text-* and *binary-based* messages.

To evaluate the effectiveness of the proposed approach, we compare P-gram with existing state-of-the-art message clustering methods, including ProDecoder [22], AuoReEngine [15], Modified Needleman-Wunsch [9], and a baseline algorithm (an NLP-style “vanilla” *n*-gram approach). Eight message data sets generated from real-world systems and protocols are used for our evaluation, including both textual and binary messages. The experimental results show that P-gram achieves more accurate clustering than any of the other methods.

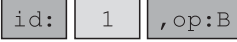
The rest of the paper is organized as follows: Section II introduces the structural features of machine-generated messages and our goals. We provide the technical details of P-gram in Section III and present the implementation details and experimental results of P-gram in Section IV. Related work is discussed in Section V. Finally, Section VI presents some concluding remarks and proposes future work.

II. MACHINE-GENERATED MESSAGES AND OUR GOAL

Machine-generated messages follow a template structure and are constructed according to message templates/formats. Given a set of messages, there may be of different *types*, where each message type follows a particular format. A message consists of fixed fields (*keywords*, that are repeated in messages of the same type) and variable fields (containing *payload* data), as shown in Figure 1. In the context of this paper, the concept of a keyword is similar to but broader than its usual meaning in machine languages. That is, a *keyword* is a maximum consecutive sequence of bytes or characters that is reserved by and has a special meaning in a machine language, which is either compulsory (occurs in all messages of a given type) or optional (occurs in only some messages of a given type). In general, the type of a message is determined by the keywords it contains.

Figure 2 gives an example list of messages from interacting with a system. It follows a fictional directory service protocol similar to the widely used Lightweight Directory Access Protocol (LDAP), but is simplified to make our running example easier to follow.

Each message shows a number of fields. Manually, we can identify the keywords, such as “id:”, “ , op:B”,



id:1,op:B
 id:2,op:S,cn:Juliet,ou=Legal,ou=Corp,c=AU
 id:3,op:S,cn:Pak,ou=Maritime,ou=Projects,c=Br
 id:4,op:S,cn:Haycne,ou=Power,ou=Team,c=IN
 id:5,op:S,cn:Han,ou=Services,ou=Customer,c=CN
 id:6,op:B

FIGURE 2. An example list of 6 messages following the fictional Lightweight Directory Access Protocol (LDAP). There are two types of messages covered in this example: op:B and op:S. The op:S messages present 5 keywords: “id”, “cn”, “ou”, “ou” and “c”. Note that, (i) “ou” is a repetitive keyword with 2 occurrences; (ii) the keyword “c” is a sub-string of the keyword “cn”; (iii) the payload `Haycne` in message 4 has the same sub-string of the keyword “cn”.

“, op:S,cn:”, “, ou=”, and “, c=”. Note that there are two types of messages, which are determined by the keywords “, op:B” and “, op:S,cn:”.

Our goal is to design a method that, given a set of machine-generated messages, separates messages into type-specific clusters as determined by their keywords. Previous clustering methods often suffer from keyword mis-identification issues. Taking the messages in Figure 2 as an example, “ou” can be identified as a keyword in existing methods, as it has a high frequency of occurrence. However, it will only be considered as having appeared in a message, ignoring its *multiple occurrences* in a message. Another example would be “cn”, which can be identified as a keyword by existing methods. However, the *short* keyword “c” will be ignored, because “c” is a sub-string of “cn” and it will be merged into “cn” by existing methods. In addition, the “cn” from *payload* “Haycne” is often wrongly treated as a keyword in the message by existing methods. In this paper, we seek to address these issues with a new keyword identification method and achieve better clustering accuracy for machine-generated messages.

III. P-GRAM CLUSTERING

In this section, we introduce our clustering method, *P-gram*, for machine-generated messages. It considers the positions of various words in messages in determining whether or not they are keywords, achieving greater accuracy in identifying keywords and message clusters. As shown in Figure 3, P-gram has four major steps: in Step 1, we identify frequent position-specific words (or *positional n-grams*) in the messages; in Step 2, we extract the position-specific keywords (or *positional keywords*) from the message words of Step 1; in Step 3, we further refine the keywords by taking into account their *positional variations*; finally in Step 4, the messages are clustered based on the identified *features*—*positional keywords*. We present these steps in turn in the following subsections.

A. POSITIONAL N-GRAM GENERATION (STEP 1)

In this step (see **Algorithm 1**), we introduce a new concept and technique, *positional n-gram*. A traditional *n-gram* is a subsequence of n elements contained in a given sequence of

at least n elements. Hence, n denotes the number of consecutive elements that are joined together. For example, the first message “id:1,op:B” in Figure 2 contains the following 4-grams (with the element being a character): “id:1”, “d:1, ”, “:1,o”, “1,op”, “, op:”, and “op:B”.

In contrast to traditional *n-grams*, we particularly focus on the position of each *n-gram* relative to the beginning of the message. Formally, we define a positional *n-gram* as follows.

Definition 1 (Positional *n-gram*): Given a set M of messages, we use $(x_0^m x_1^m \dots x_{l_m-1}^m)$ to denote the m -th message, where x_i^m is the i -th character in message m and l_m is the length of message m . We use $t_{i,n}^m = x_i^m x_{i+1}^m \dots x_{i+n-1}^m$ to denote a positional *n-gram* of length n at position i in message m .

Based the above definition, we see in Figure 2 that the positional 3-gram “id:0,3” appears at position 0 in all messages, and the positional 5-gram “, op:B_{4,5}” appears at position 4. Similarly, other keywords appear at/around fixed positions.

To identify frequent positional *n-grams*, we use the following definition to count the number of messages that contain a given positional *n-gram*.

Definition 2 (Frequency): Given a set M of messages, for an arbitrary positional *n-gram* $t_{i,n}$, we define

$$f(t_{i,n}) = |\{t_{i,n}^m | t_{i,n} = t_{i,n}^m, 1 \leq m \leq |M|\}|, \quad (1)$$

as the frequency of the positional *n-gram* $t_{i,n}$, where $|M|$ is the cardinality of set M .

Suppose that ρ is the frequency threshold for selecting candidate keywords. The following set G gives all frequent positional *n-grams*:

$$G = \{t_{i,n} | |f(t_{i,n})| \geq \rho|M|\}. \quad (2)$$

Algorithm 1 presents the details of positional *n-gram* generation for a given set M of messages. We (1) initialize an empty set $G = \emptyset$ and a minimum length $n = 1$ of positional *n-grams*; (2) break every message down into consecutive positional *n-grams* and save them in a temporary set G_n ; and (3) count the frequency for each positional *n-gram* in G_n . For an arbitrary positional *n-gram* $t_{i,n} (\in G_n)$, if its frequency $f(t_{i,n}) \geq \rho|M|$, we add $t_{i,n}$ into G . Then, we increase n by 1 and get back to step (2) until there is no positional *n-gram* with frequency greater than $\rho|M|$ identified. Set G returns all the frequent positional *n-grams*.

An illustration of Algorithm 1 is given in Figure 3. Here, the 6 messages in Figure 2 are used as an example service trace, and we set the threshold $\rho = 1/3$. After applying Algorithm 1 (*Positional n-gram Generation*), we obtain a set of frequent positional *n-grams*. For example, the 3-grams, such as “id:0,3”, “, op_{4,3}” and “cn:10,3”, are extracted as frequent positional *n-grams* as their frequencies are greater than or equal to 2 (i.e., $6 \times 1/3 = 2$); but some 3-grams, such as “d:1,1,3” and “d:2,1,3”, are not extracted as frequent positional *n-grams* as their frequencies are less than 2. Similarly, the 4-grams “, ou=_{16,4}”

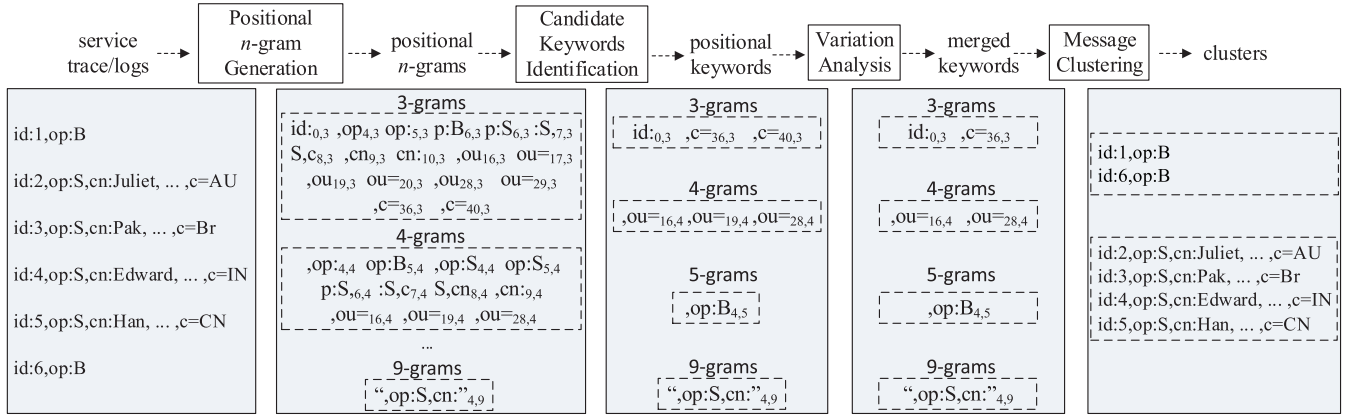


FIGURE 3. Architecture of the proposed P-gram. The messages in Figure 2 are used as an example service trace, and the results of the four main steps of P-gram working on the example trace are illustrated in each block of this figure.

Algorithm 1 Positional n -gram Generation

- 1: *Inputs:* A message set $\{(x_0^m x_1^m \dots x_{l_m-1}^m) | 1 \leq m \leq |M|\}$ and a threshold ρ .
- 2: *Output:* A set G of frequent positional n -grams and their frequencies $\{f(t_{i,n}) | t_{i,n} \in G\}$.
- 3: *Initialize:* $G = \emptyset$, and the minimum n -gram length $n = 1$.
- 4: **repeat**
- 5: Initialize an empty set $G_n = \emptyset$ to store positional n -grams of length n .
- 6: **for** ($m \in \{1, \dots, |M|\}$) **do**
- 7: Split message m into positional n -grams $P_m := \{t_{i,n}^m\}$.
- 8: **for** ($t_{i,n}^m \in P_m$) **do**
- 9: **if** ($t_{i,n} \notin G_n$) **then**
- 10: $G_n = G_n \cup \{t_{i,n}\}$.
- 11: $f(t_{i,n}) = 1$.
- 12: **else**
- 13: $f(t_{i,n}) = f(t_{i,n}) + 1$
- 14: **end if**
- 15: **end for**
- 16: **end for**
- 17: $G = G \cup \{t_{i,n} | f(t_{i,n}) \geq \rho |M|, t_{i,n} \in G_n\}$.
- 18: $f_{max} = \max\{f(t_{i,n}) | t_{i,n} \in G_n\}$.
- 19: $n = n + 1$.
- 20: **until** ($f_{max} < \rho |M|$)

with frequency 2, “, ou=19,4” with frequency 2, and “, ou=28,4” with frequency 4, are also extracted as frequent positional n -grams. In contrast, the 2-gram “cn16,2” embedded in payload Haycne in message 4 will not be extracted as an keyword, as its frequency is less than 2. In particular, the longest frequent positional n -gram extracted from the example trace is the positional 9-gram “, op:S, cn:4,9”. In the next step, we will identify positional keywords from the frequent positional n -grams extracted in Algorithm 1.

B. POSITIONAL KEYWORD IDENTIFICATION (STEP 2)

In this step (see Algorithm 2), we extract the longest common positional n -grams and significant short positional n -grams

Algorithm 2 Candidate Positional Keywords Identification

- 1: *Inputs:* Threshold ρ , a set G of frequent positional n -grams, and their frequencies $\{f(t_{i,n}) | t_{i,n} \in G\}$.
- 2: *Output:* A set S of candidate positional keywords and their independent frequencies $\{f'(t_{i,n}) | t_{i,n} \in S\}$.
- 3: **for** ($t_{i,n} \in G$) **do**
- 4: $f'(t_{i,n}) = f(t_{i,n})$.
- 5: **for** ($t_{i',n'} \in G$) **do**
- 6: **if** ($t_{i',n'} \sqsupset t_{i,n}$) **then**
- 7: $f'(t_{i,n}) = f'(t_{i,n}) - f(t_{i',n'})$.
- 8: **end if**
- 9: **end for**
- 10: **end for**
- 11: $S = \{t_{i,n} | f'(t_{i,n}) \geq \rho |M|, t_{i,n} \in G\}$.

based on the set G of frequent positional n -grams identified in Step 1.

Note that, all the positional sub-strings of a selected frequent positional n -gram in set G are also included in set G , because all these sub-strings have at least the same frequency as the enclosing positional n -gram. For example, the positional 2-gram “cn10,2” and its positional sub-string “c10,1” both have a high frequency, 4 (> 2), in Figure 2. Since “c10,1” and “cn10,2” have the same starting position, then we can calculate the frequency of the independent “c10,1” by subtracting from its own frequency the frequency of the longer positional n -grams which contain the shorter “c10,1”. Then, we get the frequency of the independent “c10,1” being 0. Hence, “c10,1” will not be a potential/candidate keyword. By using this strategy, therefore, we can extract the longest frequent positional n -grams at specific positions. Moreover, we can also keep those short positional n -grams whose independent frequency is above a given threshold. Theorems are provided below to formalize these assertions.

In the following, we present our strategy more formally. We first give the definitions of a super positional n -gram for a given positional n -gram.

Definition 3 (Super-gram): For an arbitrary positional n -gram $t_{i,n} = x_i x_{i+1} \dots x_{i+n-1}$, any positional sub-string

$t_{i',n'} = x_{i'}x_{i'+1} \cdots x_{i'+n'-1}$ (if exists) of $t_{i,n}$, where $i \leq i'$ and $i' + n' \leq i + n$, then $t_{i,n}$ is called a super-gram of $t_{i',n'}$ (denoted as $t_{i,n} \supseteq t_{i',n'}$), and $t_{i',n'}$ is called a sub-gram of $t_{i,n}$ (denoted as $t_{i',n'} \subseteq t_{i,n}$).

Based on the above definitions, we can obtain the following theorem.

Theorem 1: For an arbitrary positional n -gram $t_{i,n} \in G$, if there exists a sub-gram $t_{i',n'} \sqsubset t_{i,n}$ (i.e., $t_{i',n'} \subseteq t_{i,n}$ but $t_{i',n'} \neq t_{i,n}$), then

$$f(t_{i',n'}) \geq f(t_{i,n}), \quad \text{and, } t_{i',n'} \in G. \quad (3)$$

Theorem 1 is easy to obtain, so we do not provide the proof here. Therefore, for an arbitrary positional n -gram $t_{i,n} \in G$, any sub-gram $t_{i',n'}$ (if exists) is a frequent positional n' -gram in G .

We then give the definition of the *independent frequency* for an arbitrary positional n -gram in set G .

Definition 4 (Independent Frequency): For an arbitrary positional n -gram $t_{i,n} \in G$, we define

$$f'(t_{i,n}) = f(t_{i,n}) - \sum_{\{t_{i',n'} \in G | t_{i',n'} \sqsupset t_{i,n}\}} f(t_{i',n'}), \quad (4)$$

as the independent frequency of $t_{i,n}$.

The following theorem gives our fundamental strategy for candidate positional keywords identification.

Theorem 2: Given a positional n -gram $t_{i,n} \in S$, where S is defined as follows,

$$S = \{t_{i,n} \in G | f'(t_{i,n}) \geq \rho |M|\}, \quad (5)$$

we have,

- if $f'(t_{i,n}) = f(t_{i,n})$, \nexists a super-gram of $t_{i,n}$ in set G ;
- if $f'(t_{i,n}) < f(t_{i,n})$, \exists a super-gram $t_{i',n'} \sqsupset t_{i,n}$, such that $t_{i',n'} \in S$.

Proof: Since $t_{i,n} \in S$, we have $t_{i,n} \in G$ and $f'(t_{i,n}) \geq \rho |M|$. Based on the definition of $f'(\cdot)$, we have $f'(t_{i,n}) \leq f(t_{i,n})$. If $f'(t_{i,n}) = f(t_{i,n})$, from Eq. (4), we know that there is no super-gram of $t_{i,n}$ in set G . This proves the first result of the theorem.

If $f'(t_{i,n}) < f(t_{i,n})$, based on Eq. (4), there exist super-gram(s) $t_{i',n'}$ of $t_{i,n}$. We use $T = \{t_{i',n'} | t_{i',n'} \sqsupset t_{i,n}, t_{i',n'} \in G\}$ to denote the set of all super-grams of $t_{i,n}$, and use $t_{\tilde{i},\tilde{n}} = \arg \max_{t_{i',n'} \in T} \{|t_{i',n'}|\}$ to denote the longest sup-gram in set T . Note that, $f'(t_{\tilde{i},\tilde{n}}) = f(t_{\tilde{i},\tilde{n}})$. Therefore, we have $t_{\tilde{i},\tilde{n}} \in S$. This proves the second result of the theorem.

Therefore, for an arbitrary positional n -gram $t_{i,n} \in G$, if $f(t_{i,n}) = f'(t_{i,n})$, then $t_{i,n}$ itself is the *longest* positional n -gram for all of its sub-grams in G . Hence, $t_{i,n}$ is a potential keyword. If $f'(t_{i,n}) < f(t_{i,n})$, then there exist other longer positional n -gram(s) in G . Note that, if $f'(t_{i,n}) \geq \rho |M|$, the independent $t_{i,n}$ itself is also a potential keyword. Therefore, S returns a set of potential keywords, which can be either *longest* positional n -grams or *short* frequent independent positional n -grams.

Algorithm 2 presents the details of identifying candidate positional keywords based on Theorem 2. We initialize an

empty set $S = \emptyset$. For each positional n -gram $t_{i,n} \in G$, we first initialize its independent frequency $f'(t_{i,n})$ as the generic frequency $f(t_{i,n})$ of $t_{i,n}$. Then, we subtract $f'(t_{i,n})$ the frequencies $f(t_{i',n'})$ of the super-grams $t_{i',n'} (\sqsupset t_{i,n})$. Finally, for an arbitrary positional n -gram $t_{i,n} \in G$, if its independent frequency $f'(t_{i,n})$ is greater than $\rho |M|$, we add it into set S . Set S returns all the candidate positional keywords.

As illustrated in the *Candidate Keywords Identification* step in Figure 3, there are 8 positional keywords extracted from the frequent positional n -grams in Step 1. In particular, the positional 5-gram “, op:B4,5” contains *sub-strings*: positional 3-grams “, op4,3”, “op:5,3” and “p:B6,3”, and the positional 4-grams “, op:4,4” and “op:B5,4”. Hence, “, op:B4,5” is kept as a positional keyword, while all the sub-strings are removed. Similarly, the positional 9-gram “, op:S, cn:4,9” is kept as a positional keyword, while all of its sub-string (e.g., “, op:S4,5” and “op:S, 5,5”) are removed. The positional 4-grams “, ou=16,4”, “, ou=19,4” and “, ou=28,4” are kept as positional keywords as they are the super-grams of themselves. Note that, “, ou=” is a repetitive keyword. More specifically, “, ou=16,4” and “, ou=19,4” correspond to the first “, ou=” keyword, and “, ou=28,4” corresponds to the second “, ou=” keyword. In the next step, we introduce an approach to merge these positional keywords through analyzing their position variations.

C. MERGING POSITIONAL KEYWORDS THROUGH VARIATION ANALYSIS (STEP 3)

Note that, due to the variable length of payloads, the exact position of message keywords may vary in a certain “window” (range). Therefore, we should consider all the occurrences of a keyword at different positions of a window as one positional keyword that has the aggregate frequency at these different positions. For keywords with multiple occurrences in a message, their positions may vary in multiple windows. In this step, we analyze the variation of the positions of each keyword in set S , and merge the positional keywords within their windows, to reduce the effect of payloads with variable lengths.

Figure 4 gives an example of the position distribution of a word in messages. As we can see, the word shows high density in the intervals [0, 10) and [40, 50], but low density in the intervals [10, 20), [20, 30) and [30, 40). Intuitively, a high-density window (interval) is more likely to contain a keyword, while the low-density window may contain noise (false keywords) from payloads. To identify those high-density windows, we adopt the Parzen-Window Density Estimation [3] to estimate the *window size* of keywords.

For an arbitrary candidate keyword $t \in S$ (without position), we use $X = \{x | t_{x,n} = t, t_{x,n} \in S\}$ to denote the set of all the possible positions of t . Given a window size δ_t , we can split X into $\lceil \frac{x_{max} - x_{min}}{\delta_t} \rceil$ windows, where x_{max} and x_{min} are the maximum and minimum positions in X . For each window, say $[x, x + \delta_t)$, we can calculate the *probability density* of t

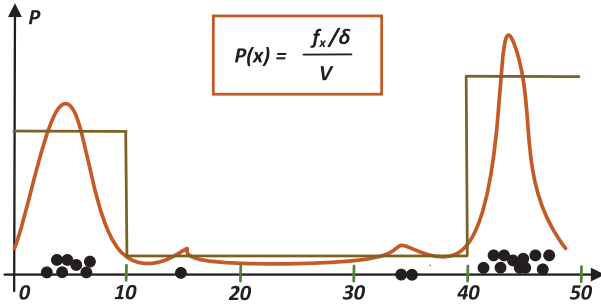


FIGURE 4. Example of the position distribution of a word in messages.

in the window as follows,

$$p(t, x) = f'_x(t) / (\delta_t \cdot V_t), \quad (6)$$

where V_t is the volume of all of t 's possible positions in X , and $f'_x(t)$ is the total independent frequency of t 's positional n -grams in the window $[x, x + \delta_t]$, i.e., $f'_x(t) = \sum_{i \in [x, x + \delta_t]} f'(t_{i,n})$. Thus, to analyze the position variation of t is to find its optimal probability density function. In other words, we need to find out the optimal window size δ_t . As suggested by [3], we set the window size as

$$\delta_t = 1.06 \cdot \sigma_t \cdot V_t^{-1/5}, \quad (7)$$

where σ_t is the standard deviation of t 's positions. Note that the smaller the standard deviation, the smaller the size of the window.

To filter out noise positions of t in X , we set a probability density threshold $\epsilon(t)$ for t . Particularly, we set $\epsilon(t)$ as half of t 's average probability density over all windows:

$$\epsilon(t) = \frac{1}{2N(t)} \cdot \frac{f'(t)}{\delta_t \cdot V_t}, \quad (8)$$

where $N(t)$ is the number of t 's position windows, and $f'(t)$ is the total independent frequency of t 's positional n -grams with positions in X . Then, for each window, say $[x, x + \delta_t]$, if $p(t, x) \geq \epsilon(t)$, we use the first positional keyword $t_{x_{min},n}$ in the window to represent all possible variations of t in the window, and aggregate the independent frequencies in the window to $t_{x_{min},n}$ as follows:

$$f'(t_{x_{min},n}) = \sum_{i \in [x, x + \delta_t]} f'(t_{i,n}), \quad (9)$$

and put $t_{x_{min},n}$ into a new set K (to store merged positional keywords); if $p(t, x) < \epsilon(t)$, we skip to the next window. Hence, we can

Algorithm 3 presents the details of merging positional keywords based on the above position variation analysis. We initialize an empty set $K = \emptyset$, and sort candidate positional keywords in S alphabetically, so that $\forall t \in S$, we can easily retrieve all the positions of t . Then, we calculate window size δ_t and the probability density threshold $\epsilon(t)$ for t . Then, we examine each window and get the probability density p of t in the window. If $p \geq \epsilon(t)$, we put the candidate positional keyword of t with the lowest position in the window into

Algorithm 3 Merge Positional Keywords via Variation Analysis

- 1: *Inputs*: A set S of candidate positional keywords and their independent frequencies $\{f'(t_{i,n}) | t_{i,n} \in S\}$.
- 2: *Output*: A set K of merged positional keywords and their windows $\Delta = \{\delta_t\}$.
- 3: *Initialize*: $K = \emptyset$, $\Delta = \emptyset$, and sort S alphabetically.
- 4: **repeat**
- 5: $t = S(0).keyword$, retrieve t 's all positions: $X = \{x | t_{x,n} = t, t_{x,n} \in S\}$.
- 6: Calculate window size δ_t using Eq. (7).
- 7: Calculate the probability density threshold $\epsilon(t)$ using Eq. (8).
- 8: Sort X in ascending order, $j = X(0)$.
- 9: **while** ($j < |X|$) **do**
- 10: **for** ($k = (j + 1) .. (|X| - 1)$) **do**
- 11: **if** ($X(k) - X(j) > \delta_t$) **then**
- 12: Get $p(t, j)$ in window $[X(j), X(k)]$ using Eq. (6).
- 13: **if** ($p(t, j) \geq \epsilon(t)$) **then**
- 14: $K = K \cup \{t_{X(j),n}\}$, $\Delta = \Delta \cup \{X(k - 1) - X(j)\}$.
- 15: $f'(t_{X(j),n}) = \sum_{i \in [X(j), X(k)]} f'(t_{i,n})$.
- 16: **end if**
- 17: **break;**
- 18: **end if**
- 19: **end for**
- 20: $j = k$.
- 21: **end while**
- 22: $S = S \setminus \{t_{x,n} | x \in X\}$.
- 23: **until** ($S = \emptyset$)

set K , record its window size, and update its independent frequency to the aggregated independent frequency of all t 's positional keywords in the window; otherwise, we skip to the next window of t . Set K returns all positional keywords.

As illustrated in the *Variability Analysis* step in Figure 3, the positional 4-grams “, ou=16,4” and “, ou=19,4” are merged to one keyword “, ou=16,4” with independent frequency 4, and the positional 4-gram “, ou=28,4” is kept as an individual keyword. Now, we will explain the detailed computation in merging the first two positional 4-grams. First, it is easy to get the volume $V = 12$ (i.e., $28 - 16$) of “, ou=”, and the standard deviation of its positions: $\sigma = 6.245$. Then, we can use Eq. (7) to calculate the window size $\delta = 4.0$, and subsequently get 6 windows for “, ou=”: [16, 20], [20, 24), [24, 28), [28, 32), [32, 36) and [36, 38]. By using Eq. (6), we obtain the probability densities for “, ou=” in these windows as: 0.0828, 0, 0, 0, 0, and 0.0828. Meanwhile, we obtain the density threshold $\epsilon = 0.0138$ using Eq. (8). Hence, we obtain two effective windows [16, 20) and [36, 38) for “, ou=”. Finally, we use “, ou=16,4” to represent the “, ou=” keyword in window [16, 20) and “, ou=28,4” to represent the “, ou=” keyword in

window [36, 38]. Similarly, the positional 3-grams “, c=36,3” and “, c=40,3” are merged to one keyword “, c=36,3” with the independent frequency updated to 4.

D. MESSAGE CLUSTERING (STEP 4)

In this step, we use the positional keywords in set K as *features* for message clustering. Hence, we can map messages into a feature vector space and apply a clustering algorithm, such as K-Medoids, on the vectors to cluster messages.

We use $|K|$ to denote the cardinality of set K . So, there are $|K|$ positional keywords (*features*) merged in Algorithm 3. Now, we can construct a $|K|$ -dimension vector for each message, say m :

$$v_m = [w(t_{i,n})]_{1 \times |K|}, \quad 1 \leq m \leq |M|, \quad (10)$$

where $w(t_{i,n})$ is the *weight* of feature $t_{i,n}$ in message m . For an arbitrary feature $t_{i,n}$, we check the existence of $t_{i,n}$ in message m by examining if $t_{i,n}$ is covered by the positional sub-string $x_i^m x_{i+1}^m \cdots x_{i+\delta_t-1}^m$ in message m , where $\delta_t \in \Delta$ is the window size of $t_{i,n}$. If $t_{i,n} \sqsubseteq x_i^m x_{i+1}^m \cdots x_{i+\delta_t-1}^m$, we need to assign a weight $w(t_{i,n})$ for $t_{i,n}$; otherwise, set $w(t_{i,n}) = 0$. In the following, we introduce an entropy-based positional weighting method to measure the weight $w(t_{i,n})$.

As some strings from payloads may be extracted as features in set K (due to their high occurrence), we wish to assign a greater weight to the structural features (such as operation type) of the message and a lower weight to noise (extracted from payload). To do so, we make use of the observation that structure features are more stable than payload data. We use entropy as a measurement of variability, and use it as the basis to calculate a weighting for each byte position. The Shannon Index entropy is adopted to measure the variability [9],

$$E_j = - \sum_{i=1}^R q_{ji} \log q_{ji}, \quad (11)$$

where E_j is the Shannon entropy for the characters at the j -th position, q_{ji} is the ratio of the i -th character in the character set of position j , and R is the total number of characters in the character set. The less diversity of characters at a position, the lower entropy that position has.

Then, we calculate the variability of a feature by adding up the entropy of the positions covered by the feature. Hence, for a feature $t_{i,n}$, its entropy is:

$$E(t_{i,n}) = \sum_{j=i}^{i+n+\delta_t-1} E_j. \quad (12)$$

To assign a high weight to stable keywords and a low weight to dynamic noise, we invert the entropy for each feature by applying a scaling function of the form given in the following *weight* equation

$$w(t_{i,n}) = \frac{1}{[1 + b \cdot E(t_{i,n})]^c}, \quad (13)$$

where b and c are positive constants. As suggested by [9], we set $b = 1$ and $c = 10$ in our experiments.

Algorithm 4 Message Clustering

- 1: *Input*: A message set $\{(x_0^m x_1^m \cdots x_{m-1}^m) | 1 \leq m \leq |M|\}$, positional keywords in set K and their window sizes in set Δ .
 - 2: *Output*: Message clusters $\{c_1, c_2, \cdots, c_L\}$.
 - 3: *Initialize*: A zero matrix $W \in \mathcal{R}^{|M| \times |K|}$.
 - 4: **for** $(t_{i,n} \in K)$ **do**
 - 5: Calculate the weight $w(t_{i,n})$ for $t_{i,n}$ from Eq. (13).
 - 6: **end for**
 - 7: **for** $(k \in \{0, \cdots, |K| - 1\})$ **do**
 - 8: $t_{i,n} \leftarrow K(k)$; get $t_{i,n}$'s window size δ_t from set Δ .
 - 9: **for** $(m \in \{1, \cdots, |M|\})$ **do**
 - 10: **if** $(t_{i,n} \sqsubseteq x_i^m x_{i+1}^m \cdots x_{i+\delta_t-1}^m)$ **then**
 - 11: Set $W(m, k) = w(t_{i,n})$.
 - 12: **end if**
 - 13: **end for**
 - 14: **end for**
 - 15: Apply K-Medoids on matrix W and obtain clusters $\{c_1, c_2, \cdots, c_L\}$.
-

Finally, we get a $|M| \times |K|$ weight matrix, $W = [w_{mk}]$, for the given message set, where, if the k -th feature appears in the m -th message, we put w_{mk} at the corresponding position in matrix W , where w_{mk} is the inverse entropy weight in Eq. (13). Then, we adopt K-Medoids to do the clustering on matrix W . **Algorithm 4** gives the details of message clustering. Please refer to the Message Clustering step in Figure 3 for an illustrative example, where the “op : B” messages and the “op : S” messages are separated correctly.

IV. EXPERIMENTAL RESULTS

To evaluate the effectiveness of the proposed P-gram, we have applied it to eight datasets of machine-generated messages from eight real-life systems. They include 3 system log files (Asgard [10], Logparser SOSP and Proxifier [12]), 2 binary protocol traces (LDAP [18] and IMS [14]), and 3 text protocol traces (LDAP, Bank SOAP [5] and Twitter REST [20]). All the datasets can be found in <https://github.com/JiaojiaoSwin/datasets>. Table 1 lists the basic statistics of these datasets. Note that, the message clusters of these datasets are imbalanced (see the Gini indexes). A lower Gini index means the message clusters are more equally distributed. IMS and SOAP present relatively low Gini indexes, while LDAP, SOSP and Proxifier present very high Gini indexes. The ratio of the smallest cluster in each dataset is also presented.

Below, we first define the evaluation metrics, then introduce the competitor techniques, and finally present the experimental results and sensitivity analysis.

A. EVALUATION METRICS FOR EFFECTIVENESS

We use two standard evaluation metrics, *Precision* and *Recall* to quantitatively evaluate and compare the effectiveness of P-gram for message clustering. The following formulas give

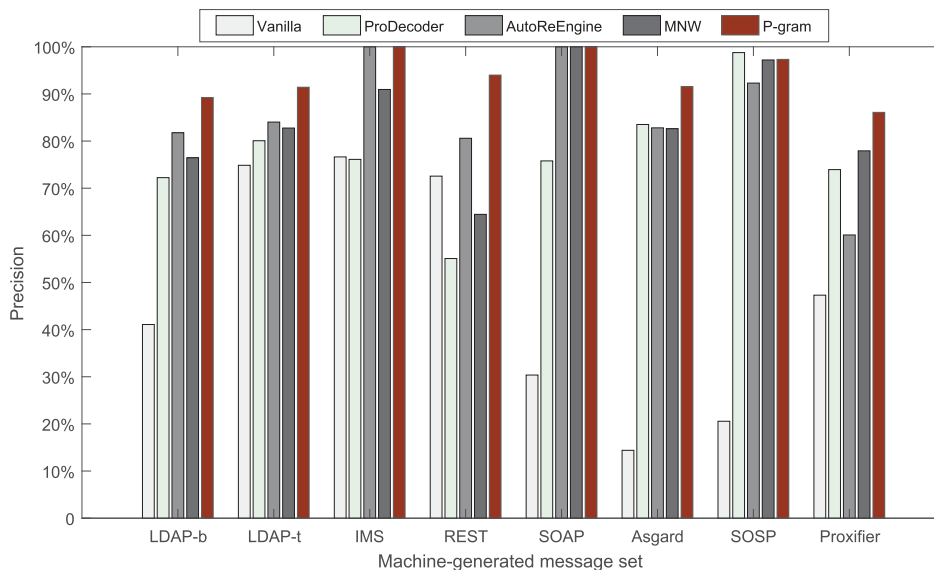


FIGURE 5. Precision of message clustering.

TABLE 1. Statistics of the datasets.

Dataset	#msg	#types	content	smallest(%)	Gini
LDAP	2181	8	binary/text	1.83	0.530
IMS	800	5	binary	12.5	0.300
SOAP	1000	6	text	14.1	0.051
REST	1825	6	text	4.05	0.455
Asgard	483	18	text	0.97	0.363
SOSP	2000	14	text	0.05	0.492
Proxifier	2000	7	text	0.05	0.649

NOTE: “smallest” stands for the message ratio of the smallest cluster.

the definitions of Precision and Recall:

$$Precision = \frac{TruePositive}{TruePositive + FalsePositive}, \quad (14)$$

$$Recall = \frac{TruePositive}{TruePositive + FalseNegative}, \quad (15)$$

TruePositive is the number of messages whose types are accurately identified. FalsePositive is the number of messages whose types are incorrectly identified as the type concerned, and FalseNegative is the number of messages whose types are not identified as the type concerned but should be. We compute these TruePositive, FalsePositive and FalseNegative using the method in [16].

B. THE COMPETITOR TECHNIQUES

We compare P-gram with two existing state-of-the-art tools (Prodecoder [22], and AutoReEngine [15]), one of our previous works (Modified Needleman-Wunsch) [9], and one baseline (i.e., Vanilla n-gram).

ProDecoder [22] identifies keywords by adopting Latent Dirichlet Allocation (LDA) models [4] used for Natural

Language Processing (NLP). Messages are then clustered according to their semantics (different combinations of keywords) using the Information Bottleneck clustering algorithm [19]. As suggested by [22], we set the number of “topics” as 40, the number of “top words” as 100, $\alpha = 0.1$, $\beta = 0.01$, and the number of iterations as 2000 in ProDecoder.

AutoReEngine [15] adapts the Apriori algorithm [1] to identify keywords. Then, the variation of keywords’ positions are calculated. Those with variations lower than a given threshold are kept as keywords, and others are filtered out as noise. Finally, keywords are sorted into vectors, and messages are clustered by the groups of keyword vectors. We set the threshold as 0.07 for keyword identification and keyword group extraction. The same threshold is used for the proposed P-gram in the keyword identification step.

Modified Needleman-Wunsch (MNW) [9] is one of our previous works. It first builds a distance matrix by calculating the entropy-weighted Needleman-Wunsch distance between pairwise messages. Then, the Visual Assessment of Tendency (VAT) clustering algorithm is applied to group messages into clusters.

Vanilla n-gram is the baseline of our work. It has been widely used in statistical NLP. In contrast to positional n-gram, the Vanilla n-gram ignores the positions of n-grams. All the other strategies of Vanilla n-gram are the same as P-gram.

C. RESULTS ON EFFECTIVENESS

In all experiments, we set $\rho = 0.07$. We set the minimum n-gram length $n = 1$ for binary messages in Algorithm 1. For text messages, as keywords are often with longer length, we set the minimum n-gram length $n = 3$; this can also save computational cost. We assume the number of clusters

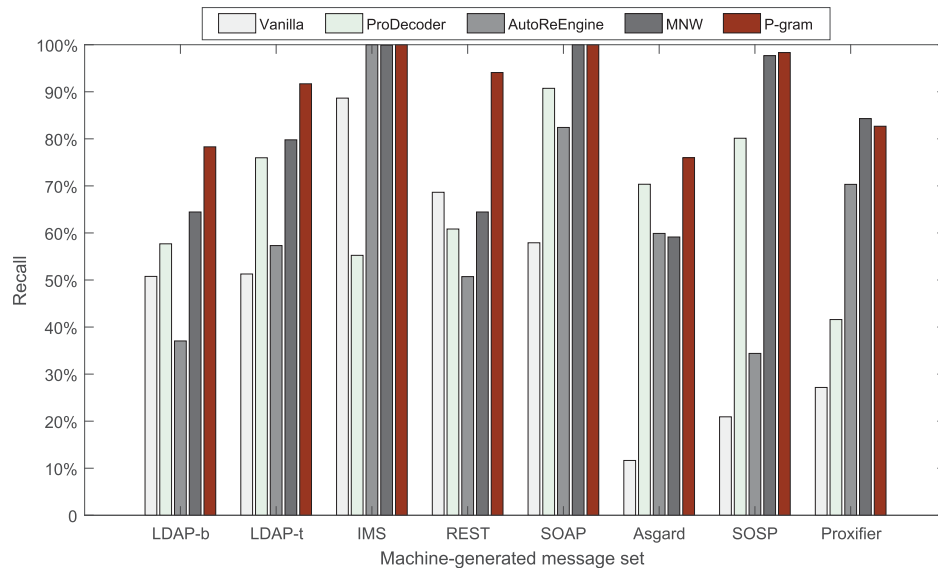


FIGURE 6. Recall of message clustering.

is prior knowledge. This is also used for all the competitor techniques except AutoReEngine, which estimates the number of message types during clustering. Figures 5 and 6 show the Precision and Recall of the clustering results of our method and other techniques. As we can see, overall, P-gram outperforms the other techniques in terms of both Precision and recall over all datasets. More specifically, it achieves 100% Precision and Recall on both IMS and SOAP datasets. It shows a relatively low Precision (87%–98%) and Recall (78%–99%) on other datasets.

We see from Figure 5 that, compared to other methods, Vanilla n -gram shows the worst performance in Precision on all datasets except on Twitter REST. This is due to the mis-identified keywords by Vanilla n -gram, which only considers the frequency of n -grams but fails to utilize the structural feature of machine-generated messages. This easily introduces noise into keywords. Hence, directly applying NLP approaches such as Vanilla n -gram on machine-generated messages fails to achieve good performance for MLP. As the Twitter REST datasets involves many natural language texts, Vanilla n -gram shows a slightly better performance than some other methods. Similar performance of Vanilla n -gram in terms of Recall can be observed in Figure 6. This, from another angle, justifies the effectiveness of our P-gram, which utilizes the metadata of machine-generated messages—the position of message keywords.

From Figures 5 and 6, we see that ProDecoder outperforms the Vanilla n -gram in terms of Precision and Recall over all datasets except IMS and Twitter REST. ProDecoder adopts the topic-model approach from NLP to extract “topics” (*i.e.*, keywords) from messages. The “topic terms” are those highly related 4-grams [22]. ProDecoder utilizes the hidden features (*i.e.*, the co-occurrence of 4-grams) of MLP.

Therefore, it achieves a better performance than Vanilla n -gram on most datasets.

AutoReEngine utilizes the variation of keywords’ positions to filter out noises from potential keywords. Overall, it shows a better performance than the previous two methods in terms of Precision (see Figure 5). However, due to its message clustering strategy (*i.e.*, messages that have the similar keyword sequence are clustered in one group) without using the prior knowledge of the number of clusters, it often generates more clusters than the ground truth clusters. Hence, as we can see in Figure 6 that, AutoReEngine shows lower Recall values than ProDecoder on some datasets, including LDAP, Twitter REST, SOAP, Asgard, and SOSP. Meanwhile, as we observed in Figures 5 and 6 that, Vanilla n -gram shows the worst performance in terms of Precision on most datasets. and it also shows poor performance in Recall on many datasets. However, on some databases, such as binary LDAP and Twitter REST, AutoReEngine shows even worse Recall than Vanilla n -gram.

The performance of the Modified Needleman-Wunsch (MNW) is close to P-gram. However, P-gram is better overall. In particular, for the IMS messages, P-gram achieves 100% Precision, but MNW has only 92% Precision (see Figure 5). For the REST messages, P-gram achieves around 93% Precision and Recall, but MNW has only around 65%. This is because MNW uses the Needleman-Wunsch distance to measure the similarity between messages. It could involve substantial noise from payloads when extracting message keywords. In contrast, P-gram effectively filters out noise by using the position information. Therefore, P-gram achieves better performance than MNW.

Furthermore, note that the proposed P-gram shows much better performance (100% Precision and Recall) on the IMS and SOAP datasets than on other datasets. This is because the

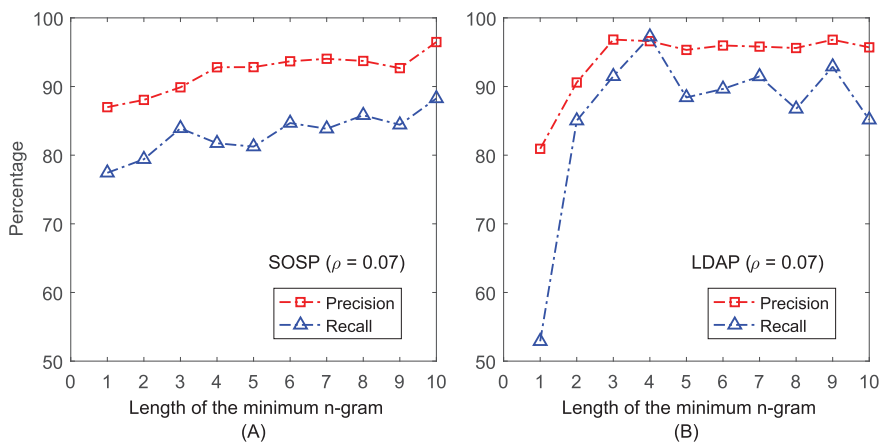


FIGURE 7. Sensitivity to the minimum length of n -grams.

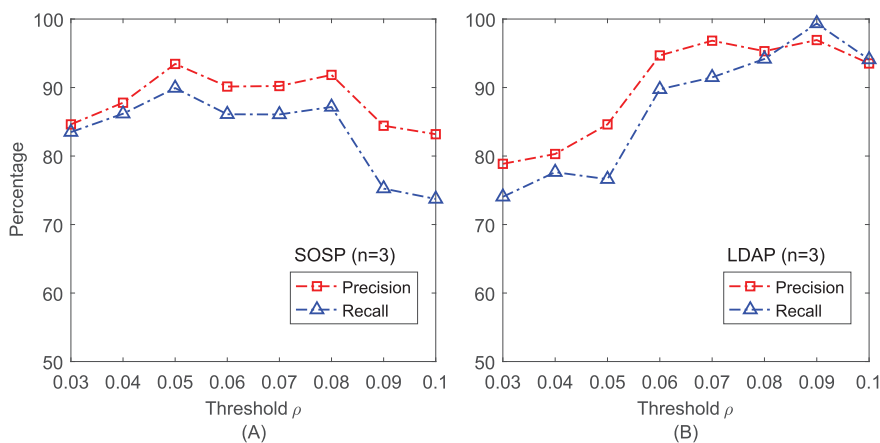


FIGURE 8. Sensitivity to the threshold ρ .

distribution of message types in these datasets are relatively even (*i.e.*, the smallest cluster accounts for 12.5% and 14.1%, respectively), while other datasets are very imbalanced (see the Gini index in Table 1). Hence, the keywords in these two datasets can be properly extracted by using the threshold $\rho = 0.07$, while the keywords for other datasets may be missed by this threshold. Later, we discuss the sensitivity of P-gram to the distribution of message types (*i.e.*, the threshold ρ) in detail.

In summary, P-gram achieves better clustering performance by addressing the keyword mis-identification problems faced by existing methods. Therefore, similar to considering semantic information in Natural Language Processing (NLP), for machine language processing (MLP), we need to consider the structural features of machine-generated messages.

D. SENSITIVITY ANALYSIS

In our experiments, P-gram involves two parameters: the threshold ρ and the minimum length n of positional n -grams. Here, we analyze the sensitivity of P-gram to these two parameters. We have chosen two datasets, SOSP logs and text LDAP protocol messages, for sensitivity analysis, as they

show uneven distribution of messages over their types (see Table 1) and hence have a greater impact on ρ and n .

In Figure 8, we fix the minimum length n to 3 and analyze the sensitivity of P-gram to ρ . In general, a very low threshold ρ would introduce substantial noise from the payloads, but a very high threshold ρ would miss keywords. As we can see in Figure 8, on the SOSP dataset, P-gram presents a relatively stable performance when $\rho \in [0.05, 0.08]$, and the Precision and Recall decreases when $\rho < 0.05$ or $\rho > 0.08$. On the LDAP dataset, P-gram presents a relatively stable performance when $\rho \in [0.06, 0.09]$, and the Precision and Recall decreases when $\rho < 0.06$ and $\rho > 0.09$. Hence, in our experiments we set $\rho = 0.07$.

In Figure 7, we fix the threshold $\rho = 0.07$ and analyze the sensitivity of P-gram to the minimum length n of n -gram. As we can see, for a fixed ρ , the Precision and Recall of P-gram increase with n . P-gram presents a low performance when the minimum length n is very low (*e.g.*, $n < 3$), especially on the text LDAP dataset. This is because, in text messages, the keywords (a maximum consecutive sequence of common bytes or characters) are often quite long. When n is very low, P-gram would introduce substantial noise from the payloads into potential keywords, resulting in message mis-clustering

and eventually low Precision and Recall of clustering performance. In our experiments, we set $n = 3$ for text messages. In addition, for binary message, we particularly set the minimum length of n -grams as $n = 1$, this is because some keywords in binary messages (such as LDAP messages) are in single bytes.

V. RELATED WORK

There are many studies that focus on the clustering of texts/messages. For example, in the area of Bioinformatics, many advanced methods [11], [26] have been proposed for the clustering of unstructured biomedical texts so as to construct the domain knowledge graph, assemble DNA sequences, etc. In the area of social media information extraction, many effective methods [17], [28] have been developed for the clustering of (generally short) user posts so as to discover trending events, identify rumors/false information, etc.

In this paper, we focus on the clustering of machine-generated messages. Many methods from Bioinformatics and social media information extraction can be applied to machine-generated messages. However, due to the specific nature of machine-generated messages (using message formats/templates), directly applying those methods cannot achieve satisfactory results [15], [25]. Hence, in the following analysis we only focus on the methods for clustering machine-generated messages.

Regarding the clustering of machine-generated messages, existing methods that are based on keyword extraction have been proven to be quite effective. They tend to split messages into keywords and other (payload) fields using n -grams [25] and/or delimiters [8]. A standard clustering technique is then performed by comparing keywords to find similar messages. In the following, we examine some representative works in keyword-based message clustering.

Cui et al. presented Discoverer [8], which uses a recursive clustering approach on tokenized messages. Messages are broken into shorter tokens based on a predefined set of delimiters. Then, tokens are compared from left to right. If two messages have the same token properties or are very similar, then those two messages are placed in a message type cluster. Finally, similar message types are merged. The technique proposed by Wang et al. [24] also uses delimiters to divide the messages into tokens. It identifies message keywords by filtering out the infrequent tokens using the Jaccard index [13]. However, the prior knowledge about delimiters used to break the messages into tokens makes these approaches inapplicable where such prior knowledge about the messages is not available. Moreover, it often splits a single message type into multiple clusters, due to the conservatism that messages in the same cluster have only very limited variations.

Wang et al. [21] proposed Biprominer targeting binary messages. It uses variable length pattern recognition to find distinguishing message keywords. It first recursively identifies frequent binary patterns of arbitrary length, called n -grams (where n denotes the number of bytes in the pattern),

in messages. Then, the probability of a keyword following another keyword is determined. Each keyword has a transition probability associated with other keywords. Finally, the messages with labeled patterns are converted into a transition probability model. Later, Wang et al. [25], [27] proposed ProDecoder, targeting both textual and binary messages by exploiting the semantics of messages. ProDecoder uses Latent Dirichlet Allocation (LDA) models taken from natural language processing to detect the n -gram keyword patterns and probable keyword sequences. Instead of using a keyword transition matrix, ProDecoder uses keyword tuples as features in an information bottleneck (IB) [19] clustering approach. IB sorts messages into clusters, with each cluster representing a different message type. Both Biprominer and ProDecoder measure the probability of each n -gram appearing in the message, and identify keywords by using a probability threshold. Then keywords are associated with specific messages for message clustering. They only consider the probability of n -grams in messages without taking into account the *template structure* of machine-generated messages, which is an essential feature of machine-generated messages and one of the main difference between machine languages and natural languages. Often, these methods put different types of messages into one cluster, leading to message mis-clustering.

Luo and Yu [15] proposed AutoReEngine. They first adapt the Apriori algorithm [1]

to find frequent n -grams as keywords in messages. Then, they filter out the “noisy” keywords by using positional variance referenced from the beginning and end of messages. The keywords that have large variation of positions are filtered out as noise. Finally, messages are clustered based on the intuition that different types of messages contain different sequences of message keywords. As AutoReEngine adopts the Apriori algorithm to extract keywords from n -grams, it often treats parts of payloads as keywords, leading to many false keywords extracted. As it only considers the frequency of n -grams when using Apriori for keywords identification, it fails to distinguish short keywords from those longer keywords that contain the short keywords as sub-strings. Meanwhile, the multiple occurrences of keywords in a message is ignored when constructing keyword sequences in the process of message clustering. Hence, messages of different types are often mixed in one cluster.

Our technique, P-gram, takes advantages of the template structure of machine-generated messages. From the above analysis, we see that existing keyword extraction based methods for message clustering faced with the keyword mis-identification issues, in particular with keyword repetition, noise (false keywords) from payloads, and the omission of short keywords covered by other longer keywords. Compared to these methods, P-gram successfully addresses these issues by considering the *position* of keywords in messages. Experiments on various textual and binary messages demonstrate the effectiveness of the proposed P-gram for clustering machine-generated messages.

VI. CONCLUSION AND FUTURE WORK

In this paper, we have proposed a novel approach, P-gram, to effectively cluster machine-generated messages. A new concept and technique, *positional n-gram*, is developed to identify message keywords. It addresses the keyword mis-identification problems suffered by existing message clustering methods. In particular, P-gram considers the positions at which message keywords appear, so as to distinguish the multiple occurrences of the same keywords in a message, filter out noise from keywords, and separate short keywords from those that contain the short keywords as sub-strings. The position-based density analysis of positional keywords further delineate keywords from the same words' occurrences in payload fields. Furthermore, we have presented theorems that prove the advantages of our proposed approach. We have demonstrated the benefits of the proposed approach by applying it to a range of machine-generated message datasets collected from real-world systems, including both textual and binary messages. The experimental results have shown the superior performance of our approach over existing state-of-the-art methods.

Based on the work in this paper, we plan to investigate general techniques to automatically extract accurate message formats from message traces. Then, we will further discover the control and data dependencies that exist in message traces. These control and data models will provide critical support for system security inspection and application behavior analysis.

Acknowledgment

J.-G. Schneider was with the School of Software and Electrical Engineering, Swinburne University of Technology, Melbourne, VIC 3122, Australia.

REFERENCES

- [1] R. Agrawal and R. Srikant, "Fast algorithms for mining association rules," in *Proc. 20th Int. Conf. Very Large Data Bases (VLDB)*, vol. 1215, 1994, pp. 487–499.
- [2] J. Antunes, N. Neves, and P. Verissimo, "Reverse engineering of protocols from network traces," in *Proc. 18th Working Conf. Reverse Eng. (WCRE)*, 2011, pp. 169–178.
- [3] G. A. Babich and O. I. Camps, "Weighted Parzen windows for pattern classification," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 18, no. 5, pp. 567–570, May 1996.
- [4] D. M. Blei, A. Y. Ng, and M. I. Jordan, "Latent Dirichlet allocation," *J. Mach. Learn. Res.*, vol. 3, pp. 993–1022, Jan. 2003.
- [5] D. Box, D. Ehnebuske, G. Kakivaya, A. Layman, N. Mendelsohn, H. F. Nielsen, S. Thatte, and D. Winer, "Simple object access protocol (SOAP) 1.1," W3C, Tech. Rep., May 2000. [Online]. Available: <http://www.w3.org/TR/SOAP/>
- [6] J. Caballero, H. Yin, Z. Liang, and D. Song, "Polyglot: Automatic extraction of protocol message format using dynamic binary analysis," in *Proc. 14th ACM Conf. Comput. Commun. Secur.*, 2007, pp. 317–329.
- [7] J. Cai, J.-Z. Luo, and F. Lei, "Analyzing network protocols of application layer using hidden semi-Markov model," *Math. Problems Eng.*, vol. 2016, Mar. 2016, Art. no. 9161723.
- [8] W. Cui, J. Kannan, and H. J. Wang, "Discoverer: Automatic protocol reverse engineering from network traces," in *Proc. USENIX Secur. Symp.*, 2007, pp. 1–14.
- [9] M. Du, S. Versteeg, J.-G. Schneider, J. Grundy, and J. Han, "From network traces to system responses: Opaquely emulating software services," 2015, *arXiv:1510.01421*. [Online]. Available: <https://arxiv.org/abs/1510.01421>
- [10] M. Farshchi, J.-G. Schneider, I. Weber, and J. Grundy, "Metric selection and anomaly detection for cloud operations using log and metric correlation analysis," *J. Syst. Softw.*, vol. 137, pp. 531–549, Mar. 2017.
- [11] L. N. L. Figueiredo, G. T. de Assis, and A. A. Ferreira, "DERIN: A data extraction method based on rendering information and N-gram," *Inf. Process. Manage.*, vol. 53, no. 5, pp. 1120–1138, 2017.
- [12] P. He, J. Zhu, S. He, J. Li, and M. R. Lyu, "An evaluation study on log parsing and its use in log mining," in *Proc. 46th Annu. IEEE/IFIP Int. Conf. Dependable Syst. Netw.*, Jun./Jul. 2016, pp. 654–661.
- [13] M. Levandowsky and D. Winter, "Distance between sets," *Nature*, vol. 234, no. 5323, p. 34, 1971.
- [14] R. Long, M. Harrington, R. Hain, and G. Nicholls, "IMS primer," IBM Int. Tech. Support Org., IBM Redbook SG24-5352-00, Jan. 2000. [Online]. Available: <http://www.redbooks.ibm.com/redbooks/pdfs/sg245352.pdf>
- [15] J.-Z. Luo and S.-Z. Yu, "Position-based automatic reverse engineering of network protocols," *J. Netw. Comput. Appl.*, vol. 36, no. 3, pp. 1070–1077, May 2013.
- [16] C. D. Manning, P. Raghavan, and H. Schütze, *Introduction to Information Retrieval*. Cambridge, U.K.: Cambridge Univ. Press, 2008.
- [17] X. Qian, M. Li, Y. Ren, and S. Jiang, "Social media based event summarization by user–text–image co-clustering," *Knowl.-Based Syst.*, vol. 164, pp. 107–121, Jan. 2019.
- [18] J. Sermersheim, *Lightweight Directory Access Protocol (LDAP): The Protocol*, document RFC 4511, 2006.
- [19] N. Tishby, "The information bottleneck method," in *Proc. 37th Annu. Allerton Conf. Commun., Control Comput.*, 1999, pp. 368–377.
- [20] (2014). *The Twitter REST API*. [Online]. Available: <https://dev.twitter.com/rest/public>
- [21] Y. Wang, X. Li, J. Meng, Y. Zhao, Z. Zhang, and L. Guo, "Biprominer: Automatic mining of binary protocol features," in *Proc. 12th Int. Conf. Parallel Distrib. Comput., Appl. Technol.*, 2011, pp. 179–184.
- [22] 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. IEEE Int. Conf. Netw. Protocols*, Oct./Nov. 2012, pp. 1–10.
- [23] Y. Wang, X. Yun, Y. Zhang, L. Chen, and G. Wu, "A nonparametric approach to the automated protocol fingerprint inference," *J. Netw. Comput. Appl.*, vol. 99, pp. 1–9, Dec. 2017.
- [24] Y. Wang, N. Zhang, Y.-M. Wu, and B.-B. Su, "Protocol specification inference based on keywords identification," in *Proc. Int. Conf. Adv. Data Mining Appl.* Berlin, Germany: Springer, 2013, pp. 443–454.
- [25] Y. Wang, Z. Zhang, D. Yao, B. Qu, and L. Guo, "Inferring protocol state machine from network traces: A probabilistic approach," in *Proc. Int. Conf. Appl. Cryptogr. Netw. Secur.* Berlin, Germany: Springer, 2011, pp. 1–18.
- [26] Z. Wang, S. Xu, and L. Zhu, "Semantic relation extraction aware of N-gram features from unstructured biomedical text," *J. Biomed. Inform.*, vol. 86, pp. 59–70, Oct. 2018.
- [27] X. Yun, Y. Wang, Y. Zhang, and Y. Zhou, "A semantics-aware approach to the automated network protocol identification," *IEEE/ACM Trans. Netw.*, vol. 24, no. 1, pp. 583–595, Feb. 2016.
- [28] S. Zenasni, E. Kergosien, M. Roche, and M. Teisseire, "Spatial information extraction from short messages," *Expert Syst. Appl.*, vol. 95, pp. 351–367, Apr. 2018.



JIAOJIAO JIANG received the Ph.D. degree in computer science from Deakin University, Australia, in 2017. She was a Postdoctoral Research Fellow with the School of Software and Electrical Engineering, Swinburne University of Technology, Australia, from 2017 to 2019. She is currently an Early Career Development Fellow with RMIT University, Melbourne, Australia. Her research interests include service virtualization and cyber security.



STEVE VERSTEEG received the Ph.D. degree in computer science from The University of Melbourne. He is currently an Adjunct Fellow with the Swinburne University of Technology. He has published more than 40 international journals and conference articles and 15 U.S. patents pending. His current research interests include service virtualization, information security, and machine learning.



JUN HAN received the Ph.D. degree in computer science from The University of Queensland, Australia. Since 2003, he has been a Full Professor of software engineering with the Swinburne University of Technology, Australia. He has published more than 250 peer-reviewed articles. His current research interests include service and cloud systems engineering, adaptive and context-aware software systems, and software architecture and quality.



MD ARAFAT HOSSAIN received the B.Sc. and M.Sc. degrees from the Rajshahi University of Engineering and Technology, Bangladesh, in 2007 and 2014, respectively. He is currently pursuing the Ph.D. degree with the School of Software and Electrical Engineering, Swinburne University of Technology, Australia. He has been working on virtualizing services to create test-bed environments for systems and services.



JEAN-GUY SCHNEIDER received the M.Sc. and Ph.D. degrees in computer science and applied mathematics from the University of Bern, Switzerland. He was a Lecturer, a Senior Lecturer, and an Associate Professor in software engineering with the Swinburne University of Technology, from 2000 to 2018. He is currently a Full Professor in software engineering with Deakin University, Burwood, VIC, Australia. His research interests include object-oriented and service-oriented systems, and scripting and composition languages.



CHRISTOPHER LECKIE is currently an Associate Professor and the Deputy-Head of the Department of Computer Science and Software Engineering, The University of Melbourne, Australia. He has over two decades of research experience in artificial intelligence (AI), especially for problems in telecommunication networking, such as data mining and intrusion detection. His research into scalable methods for data mining has made significant theoretical and practical contributions in efficiently analyzing large volumes of data in resource-constrained environments, such as wireless sensor networks.



ZEINAB FARAHMANDPOUR received the B.Sc. and M.Sc. degrees in software engineering and artificial intelligence, in 2009 and 2012, respectively. She is currently a Ph.D. Researcher in computer science with the Swinburne University of Technology, Melbourne, Australia. Her main research interests include artificial intelligence, natural language processing, software engineering, data mining, time series analyses, mathematical modeling, service virtualization, and software and network security.

...