A Comparative Study of Sequence Identification Algorithms in IoT Context

Pierre-Samuel Greau-Hamard, Moïse Djoko-Kouam, Yves Louët

A Comparative Study of Sequence Identification Algorithms in IoT Context

P. -S. Gréau-Hamard [1,2], M. Djoko-Kouam [1] and Y. Louet [2]
[1] Informatics and Telecommunications Laboratory, ECAM Rennes Louis de Broglie, Bruz, France
[2] Signal, Communication, and Embedded Electronics (SCEE) team, Institute of Electronic and Telecommunications of Rennes (IETR), CentraleSupélec, Rennes, France
E-mail: {pierre-samuel.greau-hamard, moise.djoko-kouam}@ecam-rennes.com, Yves.Louet@centralesupelec.fr

Summary: In the fast developing world of telecommunications, it may prove useful to be able to analyse any protocol one comes across, even if it is unknown. To that end, one needs to get the state machine and the frame format of the protocol. These can be extracted from network and/or execution traces via Protocol Reverse Engineering (PRE). In this paper, we aim to evaluate and compare the performance of three algorithms used as part of three different PRE systems of the literature: Aho-Corasick (AC), Variance of the Distribution of Variances (VDV), and Latent Dirichlet Allocation (LDA). In order to do so, we suggest a new meaningful metric complementary to precision and recall: the fields detection ratio. We implemented and simulated these algorithms in an Internet of Things (IoT) context, and more precisely on Zigbee Data Link Layer frames. The results obtained clearly show that the LDA algorithm outperforms AC and VDV.

Keywords: Protocol Reverse Engineering, AC, VDV, LDA, Zigbee, IoT, Data Link Layer, Performance Comparison.

1. Introduction

With the ever growing development of telecommunications, and especially Internet of Things (IoT), a lot of new protocols are constantly appearing. In order to know what they are used for, we need to understand how they work.

In this paper, we place ourselves in the context of a communicating object coming into an unknown environment and wanting to establish a communication with the existing networks. To that end, the object needs to have 'generic', or 'multi-standard' behavior, i. e. to be able to adapt itself to whichever standard is used in the target environment. It is the same goal as the one pursued by Software Defined Radio, except that in our case, we propose to learn the unknown protocol of the environment, and not just identify it from a database.

This is the goal of Protocol Reverse Engineering (PRE), a family of techniques which aims at reconstructing the frame formats and/or the state machine of a target unknown protocol through analyzing execution traces and/or network traces.

There is no precisely defined procedure to perform PRE, but the most encountered one [1] is a five-step process. (i) Firstly, the radio traffic is intercepted and the frames issued by the targeted protocol are isolated. (ii) Next, the meaningful binary sequences (features) of these frames are identified, (iii) and then the frames are grouped by format via the use of these features. (iv) Within each group, sequence alignment is performed, and, finally, (v) the frame formats and/or the state machine of the targeted protocol are reconstructed.

In this paper, we focus solely on the second step, the identification of remarkable sequences. This step aims at reducing the quantity of information needed to label a frame. This is achieved by identifying the remarkable sections of the frames, i. e. in our case by spotting the recurring sequences and their positions. Such sequences are most probably keywords. Our goal is to evaluate and compare the performance of different techniques achieving this, in order to obtain useful data for choosing a technique or a family of techniques to be used in a real-life system. To this end, we selected the Variance of the Distribution of Variances (VDV) [2], Aho-Corasick (AC) [3], and Latent Dirichlet Allocation (LDA) [4] techniques.

The simulation context in which we will simulate the performance of these techniques lies in the analysis of Data Link Layer (DLL) frames of the Zigbee protocol.

Most of the surveys in the PRE domain involve comparing a rather narrow range of tools and their approaches without delving into the exact mechanics or presenting their performance, like in [5]. However, some of them are more exhaustive, and present in detail the techniques used by the tools [6] and the protocols they are able to reverse engineer [7].

Nevertheless, these surveys do not refer to the performance of the different tools in a quantifiable way, and they also do not present the individual performance of the techniques used in each tool. Such an approach is legitimate, as they browse a wide range of PRE tools, but this is where the particularity of our paper stands. We select only three techniques as opposed to the dozens present in the previous surveys, and we evaluate their performance through simulations, which has not been done in the previous papers.

The rest of this paper is organized as follows: section 2 presents the theory related to the three techniques studied; in section 3, we simulate and compare them; and finally, we conclude in section 4.

2. State of the Art of the Three Techniques

In this section, we present the principle and mechanisms of each of the sequence identification techniques, as well as the practical algorithms designed from them to fit our context.

2.1 Variance of the Distribution of Variances

This technique aims at statistically identifying in a population the parts which offer the least variability. The following presentation is based on the approach proposed by A. Trifilò et al [2].

The VDV technique considers a population formed of groups of individuals. The latter are themselves composed of elements which can take different numerical values. The technique unfolds in five steps:

- For each group, calculating the average, then the variance of the value of each element across all the individuals in a given group.
- Across groups, calculating the average, then the variance of these variances.
- Retaining the elements whose variance of the variances is less than a given filtration threshold.

The actual algorithm used in our context derives from the technique above, with some modifications.

The groups of individuals previously considered are now replaced by flows composed of DLL frames to be analysed, and the base unit is switched from element to 'token', a *n*-bit long position slot on the frames which can assume different sequences of *n* consecutive bits.

To be able to detect fields regardless of their position, all the possible tokens obtainable from a frame are created.

The filtration threshold actually used for filtering (FT_{VDV}) is not a fixed value, but a value proportional to the average variance of the variances. The proportionality coefficient is called filtration threshold coefficient (FT^c_{VDV}), and the formula linking FT_{VDV} and FT^c_{VDV} is:

$$FT_{VDV} = \overline{V_V(i)} \times FT^c_{VDV}, \quad (1)$$

with $V_V(i)$ the variance of the variances of token i.

The frames being collected on a radio link, it is not possible to clearly identify flows, so we create them by randomly attributing frames to flows following a discrete uniform law.

To enable the detection of fields of different lengths, the algorithm is executed many times, for different values of *n*, i.e. token lengths, and all the single sequences extracted from these runs are kept for metrics computation.

2.2 Aho-Corasick

This technique was designed by A. Aho and M. Corasick in order to identify a string of characters in a text [3]. However, its use can be extended to identify any pattern composed of a sequence of elements taking values from a discrete finite space. The search is then run on a sequence of these elements whose length is superior or equal to the targeted pattern. The following presentation is based on the approach proposed by Y. Wang et al. [8].

The particularity of AC is that it is based on a state machine to optimize the processing speed.

The technique operates in two major steps:

- Constructing the state machine based on the strings to search in the text.
- Scanning the whole text character by character, and notifying, for each character, the strings ending on that character.

The actual algorithm used in our context derives from the one above, with some modifications.

The text considered in the AC technique is now replaced by the DLL trace to be analysed, and the base unit is switched from character to bit. Moreover, the strings to be identified are now all the possible *n*-bit sequences.

An occurrence counter of the sequences was added, in order to perform filtering. The sequences under a threshold FT_{AC} proportional to the average number of appearances of any sequence considering a uniform distribution are filtered out. FT_{AC} is given by:

$$FT_{AC} = \frac{n-L+1}{2^L} \times FT^c_{AC}, \quad (2)$$

with *n* the number of bits of the trace, *L* the length of the searched sequences, and FT^c_{AC} the proportionality coefficient called filtration threshold coefficient. This filtering is done to keep only the frequent enough sequences.

The sequences with a similarity level superior to a given threshold are fused. By fusion, we mean that in a group of similar enough sequences, we retain the one best representing all the other ones. We achieve that through unsupervised ascendant hierarchical clustering of the sequences, using the similarity as the distance metric, defined by:

$$\mathrm{Sim(X,Y)} = \frac{\mathrm{l(X,Y)-ed(X,Y)}}{\mathrm{l(X,Y)}}, \quad (3)$$

with *X* and *Y* representing any two sequences, *l(X,Y)* the average length of *X* and *Y*, and *ed(X,Y)* the minimal edition distance between *X* and *Y*, i. e. the minimal number of operations to apply on one of the sequences to obtain the other one. All the sequences being the same length, the average length of X and Y, *l(X,Y)*, equals those of X and Y.

To enable the detection of fields of different lengths, the algorithm is executed many times, for different values of *n*, i.e. lengths of sequences, and all the single sequences extracted from these runs are

kept for metrics computation.

2.3 Latent Dirichlet Allocation

This technique comes from the machine learning domain of Information Retrieval (IR), which aims at modeling a text mathematically, in order to extract its meaning. It was designed with the objective to identify topics from a document corpus, and to associate terms coming from a dictionary with them. However, its use can be extended to regrouping sequences of single elements taking values in a finite discrete space, from a collection of data. The following presentation is based on the approach proposed by Y. Wang et al [9].

The LDA technique is first and foremost a generative model for a corpus based on a Bayesian network; the actual implemented algorithm is deduced from it upon inference.

Let us introduce the necessary notions and parameters needed to understand the generative model and the inference based on it:

- a word *w* is an element taking value from a dictionary *v* gathering all the known vocabulary.
- a document *m* is a set of words *w*, modeled by a vector.
- a corpus *W* is a set of documents *m*, modeled by a vector.
- a term *t* is the base element of the vocabulary.
- *V* is the set of terms of the dictionary or its cardinal.
- *K* is the set of topics desired or its cardinal.
- *M* is the set of documents in the corpus or its cardinal.
- α and β are the Dirichlet prior parameters of the topics over documents and the words over topics distributions, respectively.
- ξ is the parameter of the Poisson law determining the number of words in each document.
- $\overrightarrow{\theta_m}$ is the vector characterizing the topics distribution for the document *m*. $\Theta = \{\overrightarrow{\theta_m}\}_{m=1}^{M}$ is the matrix $M \times K$ characterizing the topics distribution over the documents.
- $\overrightarrow{\varphi_k}$ is the vector characterizing the terms of *v* distribution for the topic *k*. $\Phi = \{\overrightarrow{\varphi_k}\}_{k=1}^{K}$ is the matrix $K \times V$ characterizing the terms distribution over the topics.
- N_m represents the number of words in document *m*.
- $w_{m,n}$ represents the n^{th} word of document *m*. The vector $\overrightarrow{w_m}$ represents the words of document *m*. The vector of vectors $M \times N_m$ $\overrightarrow{w}$, represents the words of the corpus.
- $z_{m,n}$ represents the topic associated to the n^{th} word of document *m*. The vector $\overrightarrow{z_m}$ represents the topics respectively attibuted to each word of document *m*. The vector of vectors $M \times N_m$ $\overrightarrow{z}$, represents the topics respectively attributed to each word of the corpus.

In LDA, we consider that a corpus is a set of documents, each of those being composed of a random number of words, where the number is drawn following a Poisson law of parameter ξ. Each of the words takes a value within the dictionary.

In the generative model, to begin, Θ *and* Φ are randomly generated following a Dirichlet law of parameters α *and* β, respectively.

Firstly, for each word to be generated of each document to be generated, the topic associated to it is randomly drawn following a multinomial law parameterized by Θ, knowing the document the word is in. Next, the value of the word is drawn from the dictionary, following a multinomial law parameterized by Φ, knowing the previously drawn topic associated with the word.

The goal of the LDA technique is to infer the words over topics and topics over documents distributions, i. e. the matrices Θ and Φ, from the corpus of documents.

These distributions are intractable, so they will be estimated through Gibbs sampling [10].

The Gibbs sampling technique comes from observing that it is impossible to simultaneously infer all the latent variables of the model (i. e. the topics). So, instead, one at a time, their distributions are inferred conditionally to all the other ones, then a new realization of the inferred distribution is drawn. When repeating this operation over all the variables a large number of times, theory shows that the realizations drawn (i. e. the sample) eventually converge towards what would be sampled from the target distribution. Then, the properties of the distribution can be statistically computed from the sample.

The actual algorithm used in our context derives from the above, with some modifications.

The documents considered in the LDA technique are replaced by the DLL frames to be analysed, so the corpus consequently becomes the DLL trace. The topics are replaced by keywords of the protocol, and the words by *n*-grams, groups of *n* consecutive bits. The *n*-grams having no natural delimiters, like spaces for words, all the possible *n*-grams obtainable from a frame are created. The terms become the different possible sequences of *n* bits.

The dictionary is composed of all the possible *n*-grams with *n* bits.

The gradient of perplexity is used as the stopping criterion of the Gibbs sampler. The perplexity [11] expresses the ability of a model to generalize to unknown data. The perplexity *P* of a learning corpus *W* is calculated as follows:

$$P(W) = e^{-\frac{\sum_{m \in M} \ln p(\overrightarrow{w_m})}{\sum_{m \in M} N_m}}, \qquad (4)$$

with *M* the set of frames from the learning DLL trace, N_m the number of *n*-grams in the DLL frame *m*, and

$$p(\overrightarrow{w_m}) = \prod_{t \in V} (\sum_{k \in K} \theta_{m,k} \varphi_{k,t})^{N_m^t}, \quad (5)$$

with *V* the set of the single *n*-grams, *K* the set of the keywords, and N_m^t the number of occurrences in document *m* of the single *n*-gram *t*.

We calculate the perplexity gradient ∇P between two consecutive time indexes *n-1* and *n* as follows:

$$\nabla P(n, n-1) = \frac{|P(n) - P(n-1)|}{P(n)}. \quad (6)$$

The sampling continues as long as the perplexity gradient is above a threshold defined as the maximal perplexity gradient divided by the number of frames in the DLL trace.

Once the matrices Θ and Φ are calculated, for each keyword, the *n*-grams with the highest appearance probabilities are selected. For that purpose, the *n*-grams are ordered by descending probabilities, then iterated through, calculating the gradient within each pair of consecutive *n*-gram probabilities. Mathematically, if we consider a keyword *k*, and its associated *n*-gram distribution vector, $\overrightarrow{\varphi_k}$, in descending order, the probability gradient ∇p_k of a *n*-gram in position $n \in [2, V]$ is:

$$\nabla p_k(n) = \frac{|\varphi_k(n) - \varphi_k(n-1)|}{\varphi_k(n)}. \quad (7)$$

The first *n*-gram is always selected, and the others are selected if their probability gradient is under the threshold defined as the maximal probability gradient.

To enable the detection of fields of different lengths, the algorithm is executed many times, for different values of *n*, i.e. *n*-gram lengths, and all the single sequences extracted from these runs are kept for metrics computation.

3. Comparative Simulations

In this section, we present the metrics (including new ones proposed in this paper) used to quantify the performance of the algorithms, as well as the parameterization for the simulations. We then discuss the results produced.

3.1. Performance Metrics

In order to define the metrics quantifying the performance of the algorithms, we introduce the notions of sequence, field, and matching condition as follows:

- **Sequence**: a sequence *s* is characterized by its length *l*, value *v*, and the set of its positions $p = \{p_i\}_{i \in \mathbb{N}}$. A sequence is then represented by $s(l, v, p)$. The list of the detected sequences $S = \{s\}$ is given by the identification algorithms.
- **Field**: a field *c* is characterized by its possible lengths *L*, characteristic values *V* (null if absent), and possible positions $P = \{P_i\}_{i \in \mathbb{N}}$. A field is then represented by $c(L, V, P)$. The list of the fields $C = \{c\}$ is obtained from the Zigbee specification. A field can be either detectable ($V \neq \emptyset$) or not detectable ($V = \emptyset$).
- **Matching condition**: the sequence $s(l, v, p)$ and the field $c(L, V, P)$ match if the following property is verified:

$$\begin{cases} l \in L, v \in V, p \cap P \neq \emptyset, if\ V \neq \emptyset \\ l \in L, p \cap P \neq \emptyset, if\ V = \emptyset \end{cases} \quad (8)$$

As metrics, we first use a tradeoff between precision and recall, the F score [12] which is the harmonic mean of the two. This metric quantifies the quality of the sequence detection, i. e. the performance of the algorithm. It is given by:

$$F = \frac{2 \times P \times R}{P + R}, \quad (9)$$

with P the precision, quantifying the part of what was correctly detected from the totality of what was detected, given by:

$$P = \frac{T_p}{T_p + F_p}, \quad (10)$$

with T_P the true positives, which are the sequences correctly detected, and F_P the false positives, which are the sequences incorrectly detected.

R is the recall, quantifying the part of what was correctly detected from what was supposed to be detected, given by:

$$R = \frac{T_p}{T_p + F_n}, \quad (11)$$

with F_n the false negatives, which are the sequences incorrectly not detected.

Each sequence counts as one true positive if its properties match at least one detectable field, it counts as one false positive in all other cases.

The number of false negatives is equal to the difference between the number of remarkable sequences across all fields and the number of true positives.

In addition to the quality of the sequence detection, we want to quantify the quality of the field detection, as it is more reprensentative of the usefulness of the algorithm.

To the best of our knowledge, a metric serving that purpose does not exist, so we introduce our own, the **fields detection ratio**. It quantifies the detected information of a field, and comes in three different forms: q_l for lengths, q_v for values, and q_p for positions.

Let us consider $S' = \{s \in S | eqn(8)\}$ the set of sequences matching at least one field, and $c_i(L_i, V_i, P_i)_{i \in I}$, a generic field, with $L_i = \{L_{ij}\}_{j \in \mathbb{N}}$,

$V_i = \{V_{ij}\}_{j\in\mathbb{N}}$, $P_i = \{P_{ij}\}_{j\in\mathbb{N}}$, and I the set of all existing fields, and $s_k(l_k, v_k, p_k)$ a generic sequence.

$$q_{l_i} = \frac{Card(L_i')}{Card(L_i)}, q_{v_i} = \frac{Card(V_i')}{Card(V_i)}, q_{p_i} = \frac{Card(P_i')}{Card(P_i)}, \quad (12)$$

are, respectively, the ratios between the number of possible lengths, values, and positions of c_i detected and the total number of possible lengths, values, and positions of c_i, with $L_i' = \{L_{ij} | \exists s_k \in S' | l_k = L_{ij}\}$, $V_i' = \{V_{ij} | \exists s_k \in S' | v_k = V_{ij}\}$, $P_i' = \{P_{ij} | \exists s_k \in S' | \exists p_k | P_{ij} \in p_k \cap P_i\}$.
For each of the previous ratios, the average over all fields is calculated as follows:

$$\mu_q = \sum_{i\in I} \frac{q_i}{Card(I)}, \quad (13)$$

with q_i representing the different ratios presented in (12).

These averages are the metrics we use for our simulations.

3.2. Simulation Context

In order to evaluate the performance of the algorithms VDV, AC, and LDA, we simulate them with a DLL trace of a protocol widespread in the IoT: Zigbee [13]. This protocol is based on the standard IEEE 802.15.4 [14], widely used in the IoT for physical and data link layers.

The DLL traces to be analysed are generated by randomly creating frames from a data frame formats base we created according to Zigbee specification. The trace generation process is run at each single simulation, so the traces analysed are never the same (although they respect the same statistical properties). The traces are then processed by the algorithms in an offline procedure.

The comparative simulation was done in two steps. (i) Observing the influence of each of the algorithms parameters by simulating them over an arbitrary but wisely chosen parameters set domain. (ii) Choosing the best performing parameter set among all the ones simulated, and comparing the performance achieved.

This method allowed us to get parameter sets yielding good performance, but not the best that could be achieved by the algorithms. This is due to the fact that we used a simple optimization approach, considering that all the parameters influence the algorithms in an independent manner.

We chose to simulate all the algorithms with a number of frames varying from 1 to 1000 with a logarithmic step, and with a maximal length of the searched sequences of 4.

For the VDV algorithm, we set the number of flows randomly generated to 10 and the filtration threshold coefficient to 1.

For the AC algorithm, we set the fusion threshold and the filtration threshold to 1.

For the LDA algorithm, we set the number of keywords to 10, the maximal perplexity gradient to 1, the maximal probability gradient to 0.1, alpha to 1, and beta to 0.0001.

Note that each curve point is calculated by averaging over 100 runs of the simulation corresponding to this point parameter set.

We limited ourselves to 1000 frame traces for processing power and memory usage limitations of our simulating hardware.

3.3. Simulation Results

We first get the F score graph of the **Figure 1**.

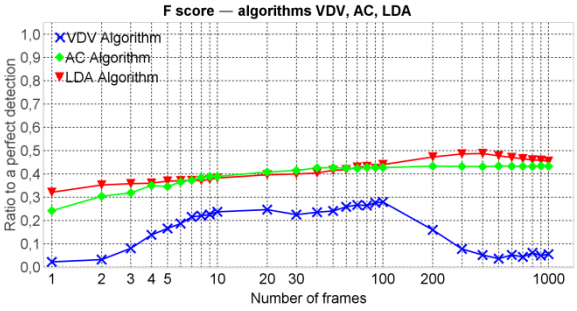

Figure 1. Best F score of the algorithms VDV, AC, and LDA

The F score synthesizing the precision and recall shows that VDV grows quickly until around 10 frames, then stabilizes around 25-30% of F score, and drops after 100 frames in the trace. On the other hand, we can observe that AC and LDA both have very near curves displaying slow linear growth, capping around 45-50% of F score, which is the best performance achieved. LDA is apparently slightly better by a small margin, but it presents a slow decrease above 400 frames.

The performances of all the algorithms grow at first because they are based on laws of large numbers, so they perform better with larger data sets to analyse.

The VDV algorithm performance drop is due to the actual filtration threshold being a value depending only on the average variance of variances, and not on the trace size, so it is appropriated only for an interval of a given number of frames. On the other hand, the AC algorithm threshold FT_{AC} depends on the number of bits in the trace, and the LDA does not use a frequency threshold to filtrate sequences, so their performances do not display a sudden drop after a certain number of frames.

The LDA slow decrease can be explained by an insufficient number of keywords, which tends to make the algorithm converge towards a smaller number of different sequences when provided with large traces, hence lower recall, so lower F score.

Let us now switch from the quality of the sequence detection to its usefulness for field detection. We draw the average fields detection ratios of the lengths, on the graph of **Figure 2**.

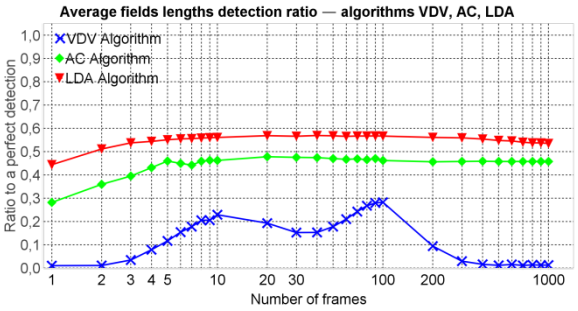

Figure 2. Best average fields lengths detection ratio of the algorithms VDV, AC, and LDA

We omit the other fields detection ratios as they present an identical behavior.

We can see that the VDV algorithm average fields length detection ratio presents the same behavior as its previous curve. The AC algorithm performance presents the same linear growth then stabilization behavior as its F score curve, while capping earlier, around 10 frames. LDA has the exact same behavior as its F score curve as well, with the decreasing phase beginning around 100 frames. It shows the best performance, with an average fields length detection ratio slightly less than 60%.

We summarize the relative performance of the three algorithms in **Table 1**. The number of stars stands, by decreasing order, for the best, average, and worst.

Table 1. Relative performance summary of the algorithms VDV, AC, and LDA

Algorithms	VDV	AC	LDA
F score	*	***	***
Average fields length detection ratio	*	**	***
Average fields values detection ratio	**	**	***
Average fields positions detection ratio	*	**	***
Total	5	9	12

We clearly see that the LDA algorithm offers the best performance, followed by the AC algorithm, and lastly the VDV algorithm.

4. Conclusion

We wanted a communicating object to be able to communicate in an unknown environment, so we needed it to learn the protocols in that environment. We chose to study and evaluate the performance of three possible sequence identification techniques which could be used in that learning procedure: VDV, AC, and LDA. To that end, we simulated them applied to the analysis of Zigbee DLL traces, and compared them.

For the purpose of comparison, in addition to the classic metric F score, we defined our own, the fields detection ratio.

From the simulations results, we can clearly state that in this context the LDA technique offers the best results, followed by the AC technique, and eventually the VDV technique.

A more powerful hardware could have allowed us to see if we could push further the performance of the LDA algorithm, and a more formal parameter optimization would have given us more precise maximal performance of the algorithms.

Nonetheless, with the results of the comparative simulation, we can now state that a technique based on Bayesian networks performs better than ones simply based on statistics or occurrences counting. This encourages us to further engage in Bayesian theory in the future.

References

[1]. O. Esoul, N. Walkinshaw, Finding clustering configurations to accurately infer packet structures from network data, in *ArXiv*, October 2016.

[2]. A. Trifilò, S. Burschka, E. Biersack, Traffic to protocol reverse engineering, in *2009 Proceedings of the IEEE Symposium on Computational Intelligence in Security and Defense Applications*, July 2009, pp. 1-8.

[3]. A. V. Aho and M. J. Corasick, Efficient string matching: An aid to bibliographic search, *Commun. ACM*, vol. 18, no. 6, June 1975, pp. 333-340.

[4]. D. M. Blei, A. Y. Ng, M. I. Jordan, Latent dirichlet allocation, *Journal of machine Learning research*, vol. 3, May 2003, pp. 993-1022.

[5]. A. Li, C. Dong, S. Tang, F. wu, C. Tian, B. Tao, H. Wang, Demodulation-free protocol identification in heterogeneous wireless networks, *Computer Communications*, vol. 55, September 2014, pp. 102-111.

[6]. S. Kleber, L. Maile, F. Kargl, Survey of protocol reverse engineering algorithms: Decomposition of tools for static traffic analysis, *IEEE Communications Surveys and Tutorials*, vol. 21, August 2018, pp. 526-561.

[7]. B. Sija, Y.-H. Goo, K.-S. Shim, H. Hasanova, M.-S. Kim, A survey of automatic protocol reverse engineering approaches, methods, and tools on the inputs and outputs view, *Security and Communication Networks*, vol. 2018, February 2018, pp. 1-17.

[8]. Y.Wang, N. Zhang, Y.-M.Wu, B.-B. Su, Y.-J. Liao, Protocol formats reverse engineering based on association rules in wireless environment, in *2013 Proceedings of the 12th IEEE International Conference on Trust, Security and Privacy in Computing and Communications*, July 2013, pp. 134-141.

[9]. Y. Wang, X. Yun, M. Shafiq, L. Wang, A. Liu, Z. Zhang, D. Yao, Y. Zhang, L. Guo, A semantics aware approach to automated reverse engineering unknown protocols, in *2012 Proceedings of the 20th IEEE International Conference on Network Protocols (ICNP)*, October 2012, pp. 1-10.

[10]. E. I. George, G. Casella, Explaining the gibbs sampler, *The American Statistician*, vol. 46, No. 3, August 1992, pp. 167-174.

[11]. G. Heinrich, Parameter estimation for text analysis, University of Leipzig, Germany, Technical Note, 2008.

[12]. C. A. Haydar, Trust-based recommender systems, Theses, Université de Lorraine, September 2014.

[13]. Zigbee Specification, ZigBee Standards Organization Std.

[14]. IEEE Std 802.15.-2003, IEEE Std.