Performance Analysis and Comparison of Sequence Identification Algorithms in IoT Context

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

Performance Analysis and Comparison of Sequence Identification Algorithms in IoT Context

P. -S. GREAU-HAMARD [1,2], M. DJOKO-KOUAM [1,2] and Y. LOUET [2]

1 Informatics and Telecommunications Laboratory, ECAM Rennes Louis de Broglie, Campus de Ker-Lann, 2 Contour Antoine de Saint-Exupéry, 35170 Bruz, France

2 Signal, Communication, and Embedded Electronics (SCEE) team, Institute of Electronic and Telecommunications of Rennes (IETR) – UMR CNRS 6164, CentraleSupélec, Campus de Rennes, Avenue de la Boulaie, 35510 Cesson-Sévigné, France

E-mail: {pierre-samuel.greau-hamard, moise.djoko-kouam}@ecam-rennes.com, Yves.Louet@centralesupelec.fr

---

Abstract: 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, 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.

This article is an extended version of the paper presented at the ASPAI' 2020 conference [8], with more detailed explanations of the compared techniques, and complementary simulation results.

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. **Fig. 1** illustrates this assumed data structuring.

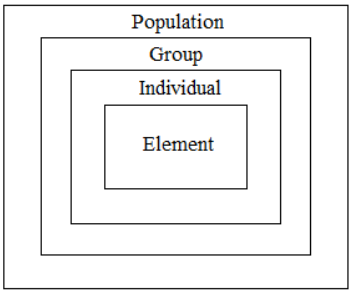

Fig. 1. Structuring of the data (=population) considered by the VDV technique

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.

Algorithm 1 in **Fig. 2** presents this process under the form of a pseudo-code algorithm.

```
Algorithm 1: VDV algorithm
  Data: Data, filtration threshold
  Result: Invariant elements positions
1 for each group do
2 |  for each element do
3 |  |  Compute the considered element average value
4 |  |  across individuals within the considered group;
5 |  end
6 end
7 for each group do
8 |  for each element do
9 |  |  Compute the considered element values variance
10|  |  across individuals within the considered group;
11|  end
12 end
13 for each element do
14|  Compute the considered element average variance
15|  across groups;
16 end
17 for each element do
18|  Compute the considered element variance of the variances
19|  across groups;
20|  if Inferior to filtration threshold then
21|  |  Select element;
22|  end
23 end
```

Fig. 2. VDV pseudo-code algorithm

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. These equivalences are summarized in **Table 1**.

Table 1. Summary of equivalent terms for the VDV algorithm

General term used in the technique explanation	Equivalent term used in our study case
Population	Data Link layer trace
Group	Flow
Individual	Frame
Element	Token

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.

The successive steps of the process are illustrated in **Fig. 3**.

The parameters of the algorithm are as follows:

- Maximal length of the sequences searched (token length)
- Filtration threshold value
- Number of generated flows

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. [9].

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

This machine is represented as a tree, with each of its states described by the character to be read to access this particular state. This automaton is ruled by three functions :

- A transition function g, defining the next state of the machine, depending on the current state and the character read, when the latter is relevant in regards to the strings searched
- A failure function f, defining the next state of the machine, depending on the current state, when the character read is not relevant in regards to the strings searched
- An activity function o, defining the word(s) finishing on a state of the machine, depending on the current state

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,

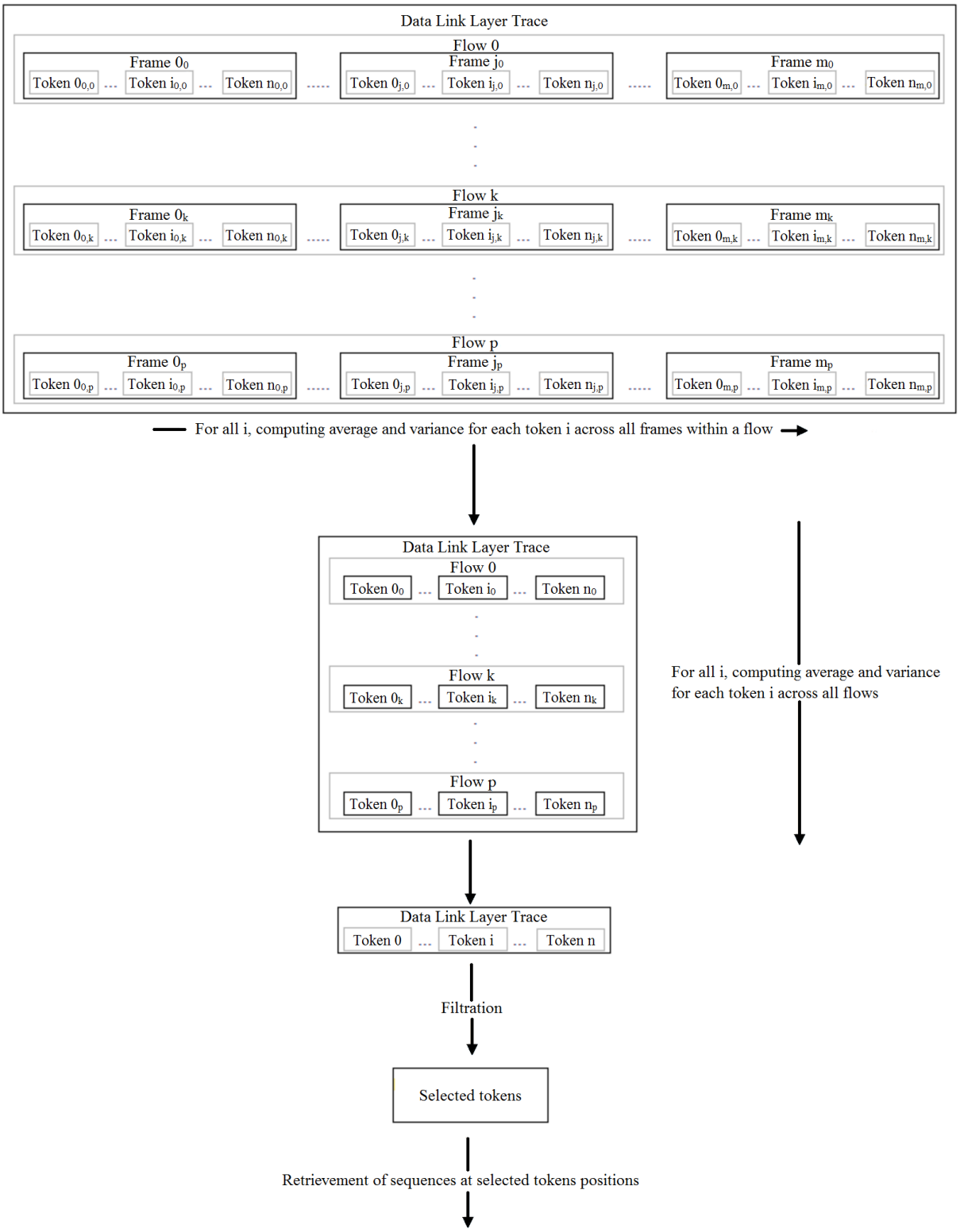

Fig. 3. Illustration of the VDV algorithm applied to our study case

the strings ending on that character.

Algorithm 2 in **Fig. 4** presents this process under the form of a pseudo-code algorithm.

Fig. 5 illustrates a sample state machine, built for searching the following strings: pit, it, pity, and paw.

Each circle represents a state. The letter written on each state corresponds to the one to be read to access this state. The empty state is the root of the tree. The thick straight lines materialize the transitions ruled by function g. They are traveled through only in the root to leaves direction. The thin curved lines materialize the transitions ruled by function f. They are traveled through only in the leaves to root direction. The thicker circles on some

```
Algorithm 2: AC algorithm
  Data: text,strings
  Result: searched strings positions
1 All elements of o are initialized to 0;
2 All elements of g and f are initialized to -1;
3 for every string do
4 |  Generate the corresponding branch of the tree with g;
5 |  Associate the state correspondig to last character
6 |  of the string to the completed string with o;
7 end
8 for each character of the dictionary do
9 |  With g, link the root with itself
10 |  for all characters not starting a string;
11 |  for each state associated to this character do
12 |  |  Insert in a queue the state
13 |  |  if it corresponds to a string begining;
14 |  |  if the state do not have any next step according to g then
15 |  |  |  if no other state corresponds to this character then
16 |  |  |  |  Link the state to root with f;
17 |  |  |  else
18 |  |  |  |  Link with f the state to one of the other
19 |  |  |  |  states corresponding to this character;
20 |  |  |  |  Repeat the linking with f until all states
21 |  |  |  |  corresponding to this character forms a chain,
22 |  |  |  |  and finally link it to root with f;
23 |  |  |  end
24 |  |  end
25 |  end
26 end
27 while the queue is not empty do
28 |  Take first element of the queue;
29 |  for each character of the dictionary do
30 |  |  if there is a corresponding state on a tree branch then
31 |  |  |  Find the deepest state which associated string
32 |  |  |  is a substring of the string associated to the current state;
33 |  |  |  Associate to the found state, in o,
34 |  |  |  the string completed on the next state of the branch;
35 |  |  |  Introduce the next state of the branch in the queue;
36 |  |  end
37 |  end
38 end
39 Start at the root;
40 for each character of the text do
41 |  while there is no next state with g do
42 |  |  Use f to move to next state;
43 |  end
44 |  Use g to move to next state;
45 |  if a string finishes on the current state then
46 |  |  Get its ending position in the text,
47 |  |  and deduce its starting position;
48 |  end
49 end
```

Fig. 4. AC pseudo-code algorithm

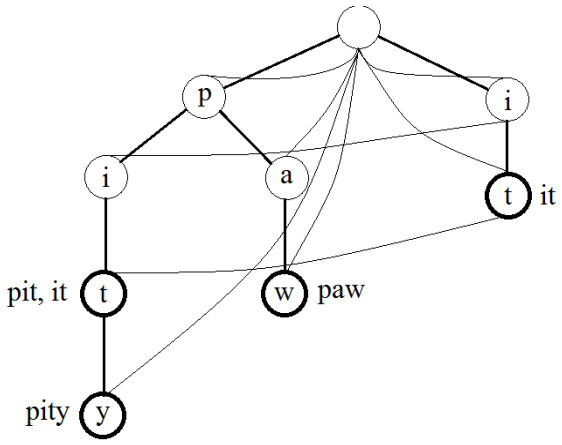

Fig. 5. Illustration of an AC technique example state machine

states mean that one or multiple strings, specified next to the states, end on these states.

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. These equivalences are summarized in **Table 2**.

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:

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

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

Table 2. Summary of equivalent terms for the AC algorithm

General term used in the technique explanation	Equivalent term used in our study case
Text	Data Link layer trace
String	Flow
Character	Frame

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:

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

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.

The parameters of the algorithm are as follows:

- Maximal length of the sequences searched

- Filtration threshold value
- Fusion threshold value

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 latent topics from a document corpus, and to associate terms coming from a dictionary to 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 [10].

The name Latent Dirichlet Allocation stems from the fact that the model uses Dirichlet as the a priori law of the latent variables (topics) allocated to the observed variables (words). Practically, the Dirichlet law is used to generate a probability vector characteristic of a multinomial law, from a vector of concentration parameters. These parameters have values varying from 0 to +∞, 0 meaning that probabilities of the generated probability vector will be 0 or 1 (before normalization) and +∞ meaning probabilities will be equal to 0.5 (before normalization). 1 means that the a priori is in fact that there is no a priori, i. e. probabilities can take any value between 0 and 1 with equal probability. The Dirichlet law was chosen for its conjugacy property with the multinomial law which is used for classification (like LDA). Conjugacy of Dirichlet law and multinomial law means that if the latent variables follow a Dirichlet a priori law, their a posteriori law will be multinomial.

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.

Let us illustrate the structures of the vectors of vectors $\overrightarrow{w}$ and $\overrightarrow{z}$ with a random example corpus and three latent subjects considered. We precise that $\overrightarrow{z}$ is filled-in with random topics ids which do not reflect what LDA would really do (outside random initialization). We consider the three following documents in the example corpus:

- Document 1 : This is an example
- Document 2 : The documents do not have all the same number of words
- Document 3 : The matrices are vectors of vectors

$$\overrightarrow{w} = \begin{bmatrix} \text{This} & \text{The} & \text{The} \\ \text{is} & \text{documents} & \text{matrices} \\ \text{an} & \text{do} & \text{are} \\ \text{example} & \text{not} & \text{vectors} \\ & \text{have} & \text{of} \\ & \text{all} & \text{vectors} \\ & \text{the} & \\ & \text{same} & \\ & \text{number} & \\ & \text{of} & \\ & \text{words} & \end{bmatrix}$$

$$\overrightarrow{z} = \begin{bmatrix} 3 & \\ 2 & \\ 2 & \\ 1 & 3 \\ 3 & 2 \\ 1 & 3 \\ 2 & 1 \\ 2 & 3 \\ 1 & 1 \\ 1 & 2 \\ & 1 \\ & 2 \\ & 1 \\ & 3 \end{bmatrix}$$

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 generative process of the documents according to LDA is described in Algorithm 3 in **Fig. 6**, and summarized on the plate diagram in **Fig. 7**.

The arrows represent the causality links

Algorithm 3: LDA generative model

Data: M,K,v,α,β,ξ
Result: $\vec{w}$

for $k = 1$ *to* K **do**
  Generate $\vec{\phi_k}$ according to a Dirichlet law of parameter $\vec{\beta}$:
  $\vec{\phi_k} \sim$ Dirichlet($\vec{\beta}$);
end
for $m = 1$ *to* M **do**
  Generate $\vec{\theta_m}$ according to a Dirichlet law of parameter $\vec{\alpha}$:
  $\vec{\theta_m} \sim$ Dirichlet($\vec{\alpha}$);
  Randomly draw a number of words according to a Poisson law of parameter ξ : $N_m \sim$ Poisson(ξ);
  for $n = 1$ *to* N_m **do**
    Randomly draw a topic $z_{m,n}$ according to a multinomial law of parameter $\vec{\theta_m}$: $z_{m,n} \sim$ multinomial($\vec{\theta_m}$);
    Randomly draw a word $w_{m,n}$ in v according to a multinomial law of parameter $\vec{\phi_{z_{m,n}}}$: $w_{m,n} \sim$ multinomial($\vec{\phi_{z_{m,n}}}$);
  end
end

Fig. 6. LDA pseudo-code generative algorithm

(probability laws) linking the random variables, represented as circles. The zones delimited by dashed rectangles materialize the different sets of elements composing the corpus:

- The topics k

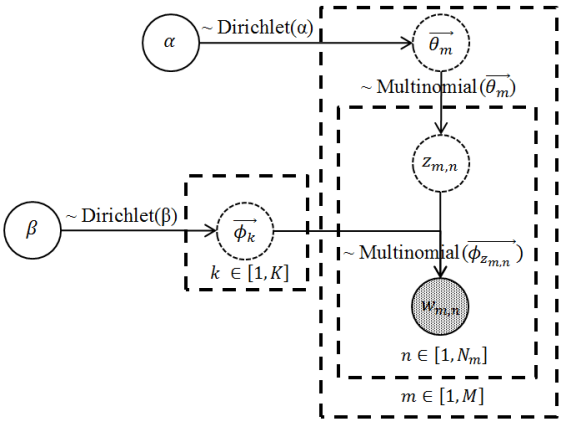

Fig. 7. Plate diagram of the LDA generative model

- The documents m
- The words n

The dashed circles correspond to the latent variables, or hidden variables, in opposition to the observed and known variables. The hatched circle is the final, observed variables generated by the model: the words present in the corpus.

It is to be noted that α and β are matrices, and ξ is a vector, but in the LDA generative algorithm, they are expressed as scalars, as we consider all the elements within each of them to be equal. When the actual α, β, and ξ are used, they are generated by replicating the scalar values as many times as needed.

Moreover, ξ does not have any use outside the generative algorithm, so it will not be mentionned anymore.

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

Given the parameters α and β, and a document m, the joint probability of all the observed variables ($\overrightarrow{w_m}$) and hidden variables ($\overrightarrow{z_m}$, $\overrightarrow{\theta_m}$, Φ) concerned by m is given by:

$$p(\overrightarrow{w_m}, \overrightarrow{z_m}, \overrightarrow{\theta_m}, \Phi|\alpha, \beta) = \prod_{n=1}^{N_m} p(w_{m,n}|\overrightarrow{\Phi_{z_{m,n}}}) p(z_{m,n}|\overrightarrow{\theta_m}) p(\overrightarrow{\theta_m}|\alpha) p(\Phi|\beta), \quad (4)$$

where $\overrightarrow{\Phi_{z_{m,n}}}$ represents the probability vector of the matrix Φ associated to $z_{m,n}$, the topic associated to the nth word of document m.

The joint probability of $\overrightarrow{w_m}$ and $\overrightarrow{z_m}$ is obtained by integrating over $\overrightarrow{\theta_m}$ and Φ:

$$p(\overrightarrow{w_m}, \overrightarrow{z_m}|\alpha, \beta) = \iint p(\overrightarrow{\theta_m}|\alpha) p(\Phi|\beta) \prod_{n=1}^{N_m} p(w_{m,n}|\overrightarrow{\Phi_{z_{m,n}}}) p(z_{m,n}|\overrightarrow{\theta_m}) \, d\Phi d\overrightarrow{\theta_m}. \quad (5)$$

The marginal distribution of $\overrightarrow{w_m}$ is obtained by summing $p(\overrightarrow{w_m}, \overrightarrow{z_m}|\alpha, \beta)$ over all the possible values of $z_{m,n}$, i. e. all the topics K:

$$
\begin{aligned}
p(\overrightarrow{w_m}|\alpha,\beta) &= \iint p(\overrightarrow{\theta_m}|\alpha)p(\Phi|\beta)\prod_{n=1}^{N_m}\sum_{z_{m,n}} p\left(w_{m,n}|\overrightarrow{\Phi_{z_{m,n}}}\right)p\left(z_{m,n}|\overrightarrow{\theta_m}\right)\, d\Phi \mathrm{d}\overrightarrow{\theta_m} \\
&= \iint p(\overrightarrow{\theta_m}|\alpha)p(\Phi|\beta)\prod_{n=1}^{N_m}\sum_{z_{m,n}} p\left(w_{m,n}|\overrightarrow{\theta_m},\Phi\right)\, d\Phi \mathrm{d}\overrightarrow{\theta_m}
\end{aligned}
\tag{6}
$$

The joint probability of all words in the corpus and their associated topics is given by:

$$p(\vec{w},\vec{z}|\alpha,\beta) = \prod_{m=1}^{M} p(\overrightarrow{w_m},\overrightarrow{z_m}|\alpha,\beta). \tag{7}$$

The marginal probability of all the words in the corpus is given by:

$$p(\vec{w}|\alpha,\beta) = \prod_{m=1}^{M} p(\overrightarrow{w_m}|\alpha,\beta). \tag{8}$$

We aim at computing the a posteriori distributions of $\vec{z}$, Φ, and Θ, given the words $\vec{w}$ observed. However, Φ and Θ can be computed directly from the distribution of $\vec{z}$, so we consider only the latter. The target distribution is consequently as follows:

$$
\begin{aligned}
p(\vec{z}|\vec{w},\alpha,\beta) &= \frac{p(\vec{w},\vec{z}|\alpha,\beta)}{p(\vec{w}|\alpha,\beta)} \\
&= \frac{\prod_{m=1}^{M} p(\overrightarrow{w_m},\overrightarrow{z_m}|\alpha,\beta)}{\prod_{m=1}^{M}\sum_{z_{m,n}} p(\overrightarrow{w_m},\overrightarrow{z_m}|\alpha,\beta)}
\end{aligned}
\tag{9}
$$

This distribution is intractable because of its denominator being a sum over K^M terms. It will therefore be estimated through Gibbs sampling [11].

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). Random initialization. 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.

Applied to the LDA, Gibbs sampling unfolds in the following two steps:

- The initialization, where the words are randomly given associated topics
- An iterative process during which, for each word $w_{m,n}$, the probability to get each of the topics for this word given the topics attributions to all the other words of the corpus is computed. The obtained probability vector parameterizes a multinomial law, which is used to draw a new topic for the word $w_{m,n}$. This step is repeated until a stopping criterion is reached.

We can then compute the terms over topics and topics over documents distributions, i. e. the matrices Φ and Θ, from the vectors of vectors $\vec{w}$ and $\vec{z}$. We can finally make a decision on which words to associate to which topics and which topics to associate to which documents based on Φ and Θ.

Fig. 8 illustrates the principle of Gibbs sampling applied to LDA. To that end, we define the following terms:

- The vector of vectors of the topics associated to the words, $\vec{z} = \left\{z_{i,j}\right\}_{1\le i\le M, 1\le j\le n(i)}$, with M the number of documents and n(i) the number of words of document i
- The vector of vectors of the topics associated to the words to which the word $w_{i,j}$ is excluded, $\overrightarrow{z_{-ij}} = \left\{z_{m,p}\right\}_{1\le m\le M, 1\le p\le n(i), m\ne i, p\ne j}$
- The vector of vectors representing the corpus, $\overrightarrow{w} = \left\{w_{i,j}\right\}_{1\le i\le M, 1\le j\le n(i)}$

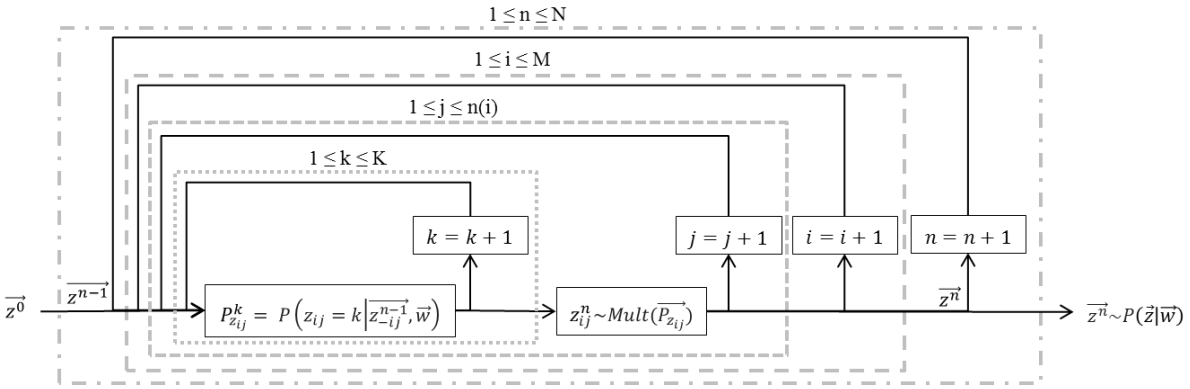

Fig. 8. Algorigram of the Gibbs sampler procedure

- The vector parameterizing the multinomial distribution of the topic $z_{i,j}$, associated to the word $w_{i,j}$, $\overrightarrow{P_{z_{i,j}}} = \left\{P^k_{z_{i,j}}\right\}_{1 \le k \le K}$
- N, the number of iterations of the algorithm (a possible stopping criterion)

The perplexity [12] expresses the ability of a model to generalize to unknown data. It is defined as the reciprocal of the geometric mean by element of a test corpus likelihood, given the model with its current parameters (which change at each learning iteration).

More practically, in the case of the LDA, it is the inverse of the average likelihood by word of the test corpus given the current state of the matrices Φ and Θ. The perplexity is expressed as follows :

$$perplexity(W_{test}) = \prod_{m \in M} p(\overrightarrow{w_m})^{-\frac{1}{N}}. \quad (10)$$

The information conveyed by the perplexity on the quality of the learned model can be interpreted as follows: for a given word slot, the probability to obtain with this model the word actually present in the test corpus will be the inverse of the perplexity. This probability is to be compared to that of a draw from a uniform law over all the words of the dictionary.

We will now present in Algorithm 4 in **Fig. 9** the learning procedure of the LDA model via Gibbs sampling, leading to the obtention of $\vec{z}$, Θ, and Φ, given the observed corpus W, the number of keywords assumed K, the dictionary v, and the Dirichlet a priori parameters α and β.

The notation –i means all elements of the considered ensemble, except the one in position i.

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. These equivalences are summarized in **Table 3**.

Table 3. Summary of equivalent terms for the LDA algorithm

General term used in the technique explanation	Equivalent term used in our study case
Corpus	Data Link layer trace
Document	Frame
Topic	Keyword
Word	N-gram
Term	N-bits sequence

Algorithm 4: LDA learning algorithm via Gibbs sampling

Data: W,K,v,α,β
Result: $\vec{z}$, Φ, and Θ

Initialize to 0 $n_m^{(k)}$, representing the number of time
a word of document m is assigned to topic k;
Initialize to 0 n_m, representing the number of words in document m;
Initialize to 0 $n_k^{(t)}$, representing the number of time
the term t is assigned to topic k;
Initialize to 0 n_k, representing the number of time
a word is assigned to topic k;
for *m = 1 to M* **do**
  for *n = 1 to* N_m **do**
    $z_{m,n}$ = k ~ multinomial(1/K);
    $n_m^{(k)} + 1$, $n_m + 1$, $n_k^{(t)} + 1$, $n_k + 1$;
  end
end
while *stopping criterion not verified* **do**
  for *m = 1 to M* **do**
    for *n = 1 to* N_m **do**
      The word $w_{m,n}$ is a term currently assigned to topic k;
      $n_m^{(k)} = n_m^{(k)} - 1$;
      $n_m = n_m - 1$;
      $n_k^{(t)} = n_k^{(t)} - 1$;
      $n_k = n_k - 1$;
      for *i = 1 to K* **do**
        Compute probability
        $$p(z_i|\vec{z_{-i}}, \vec{w}) \propto \frac{n_{k,-i}^{(t)} + \beta}{\sum_{t=1}^{V} n_{k,-i}^{(t)} + V \times \beta} \times \frac{n_{m,-i}^{(k)} + \alpha}{\sum_{k=1}^{K} n_{m,-i}^{(k)} + K \times \alpha};$$
      end
      Normalise these probabilities : $p(z_i|\vec{z_{-i}}, \vec{w}) = \frac{p(z_i|\vec{z_{-i}}, \vec{w})}{\sum_{t=1}^{K} p(z_i|\vec{z_{-i}}, \vec{w})}$;
      Draw randomly a new topic : k' ~ multinomial(p($z_i|\vec{z_{-i}}, \vec{w}$));
      Assign the word $w_{m,n}$ to topic k';
      $n_m^{(k')} = n_m^{(k')} + 1$;
      $n_m = n_m + 1$;
      $n_{k'}^{(t)} = n_{k'}^{(t)} + 1$;
      $n_{k'} = n_{k'} + 1$;
    end
  end
end
Compute Φ, with $\phi_{k,t} = \frac{n_k^{(t)} + \beta}{\sum_{t=1}^{V} n_k^{(t)} + V \times \beta}$;
Compute Θ, with $\theta_{m,k} = \frac{n_m^{(k)} + \alpha}{\sum_{k=1}^{K} n_m^{(k)} + K \times \alpha}$;

Fig. 9. LDA pseudo-code learning algorithm via Gibbs sampling

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

The gradient of perplexity between two iterations is used as the stopping criterion of the Gibbs sampler. The perplexity P of a learning corpus W is practically calculated as follows:

$$P(W) = e^{-\frac{\sum_{m \in M} \ln p(\overrightarrow{w_m})}{\sum_{m \in M} N_m}}, \quad (11)$$

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

$$p(\overrightarrow{w_m}) = \prod_{t \in V} (\sum_{k \in K} \theta_{m,k} \varphi_{k,t})^{N_m^t}, \quad (12)$$

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:

$$\nabla P(n, n-1) = \frac{|P(n)-P(n-1)|}{P(n)}. \quad (13)$$

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:

$$\nabla p_k(n) = \frac{|\varphi_k(n)-\varphi_k(n-1)|}{\varphi_k(n)}. \quad (14)$$

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.

The parameters of the algorithm are as follows:

- Maximal length of the sequences searched (n-grams length)
- Number of latent keywords assumed
- Maximal perplexity gradient value
- Maximal probability gradient value
- α a priori Dirichlet parameter
- β a priori Dirichlet parameter

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:

$$\begin{cases} l \in L, v \in V, p \cap P \neq \emptyset, if\ V \neq \emptyset \\ l \in L, p \cap P \neq \emptyset, if\ V = \emptyset \end{cases} \quad (15)$$

As metrics, we first use a tradeoff between precision and recall, the F score [13] 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:

$$F = \frac{2 \times P \times R}{P + R}, \quad (16)$$

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

$$P = \frac{T_p}{T_p + F_p}, \quad (17)$$

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:

$$R = \frac{T_p}{T_p + F_n}, \quad (18)$$

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 $S' = \{s \in S | eqn(15)\}$ 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.

$$q_{l_i} = \frac{Card(L_i')}{Card(L_i)}, q_{v_i} = \frac{Card(V_i')}{Card(V_i)}, q_{p_i} = \frac{Card(P_i')}{Card(P_i)}, \quad (19)$$

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:

$$\mu_q = \sum_{i \in I} \frac{q_i}{Card(I)}, \quad (20)$$

with q_i representing the different ratios presented in (19).

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 [14]. This protocol is based on the standard IEEE 802.15.4 [15], 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 to see the impact of the trace size on the performances, with a logarithmic increment to cover the domain with fewer points. We limited ourselves to 1000 frame traces for processing power and memory usage limitations of our simulating hardware.

We know that most of the sequences to be found in the test datasets have a length inferior or equal to 8 bits. We also observed a steep increase in the processing time when looking for up to 16-bit sequences, the longest actually present in the datasets. Therefore, the maximal length of the sequences searched will vary from 1 to 8 bits by power of two increments.

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

The metrics presented in the results graphs will be the following ones:

- The precision of the detected sequences
- The recall of the detected sequences
- The F score of the detected sequences
- The average fields lengths detection ratio (only for the comparison of the best performances)
- The average fields values detection ratio (only for the comparison of the best performances)
- The average fields positions detection ratio (only for the comparison of the best performances)

3.3. Study of VDV parameters influence

We first run our implementation of the VDV algorithm with a series of parameter sets to study their influence.

Table 4 summarizes the parameters used for the simulations of the VDV algorithm.

Table 4. Summary of the VDV algorithm simulation parameters

Parameters	Sequences length influence	Flows number influence	Filtration threshold influence
Number of frames	1 to 1000, logarithmic increment	1 to 1000, logarithmic increment	1 to 1000, logarithmic increment
Sequences length	1 to 8, power of 2 increment	8	8
Number of flows	10	5 to 30, increment of 5	10
Filtration threshold	1	1	0.0001 to 1, power of 10 increment

In the graph **Fig. 10**, we can see first of all that the precision of the VDV algorithm is inversely proportional to the maximal length of the sequences searched. This can be explained by the fact that the longer the sequences searched are, the more different sequences can be detected, and therefore the lower their respective numbers of appearances are, so the statistical thresholding is less efficient, hence the increase in false positives.

We can also a notice a rapid increase in precision with the number of frames in the DLL trace, when the latter is low (below 10 frames). This behaviour is to be expected, as the algorithm uses statistical filtering, which only makes sense on large samples. However, there is also a collapse in precision when the trace becomes large (above 100 frames). This is due to the fact that as the size of the trace increases, the variance decreases, so it becomes more difficult to discriminate the sequences to be detected with the filtration threshold, resulting in an increase in false positives and a decrease in true positives.

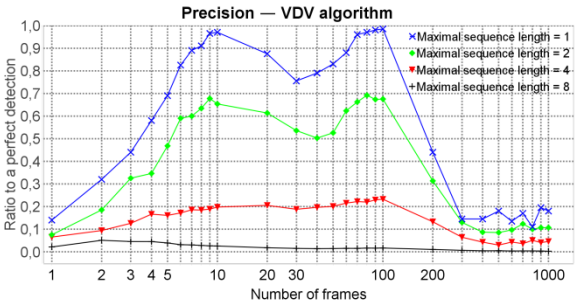

Fig. 10. Precision of the VDV algorithm relative to the trace size and parameterized by maximal sequence length

It can be seen from the graph in **Fig. 11** that the recall of the VDV algorithm is proportional to the maximal length of the the sequences searched. This seems logical, since as the length increases, it becomes possible to detect more remarkable sequences present in the DLL trace.

There is also a similar behaviour to that of the precision in relation to the number of frames in the trace, which can be explained by the same reasons.

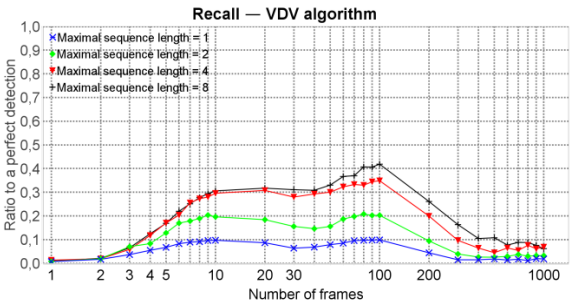

Fig. 11. Recall of the VDV algorithm relative to the trace size and parameterized by maximal sequence length

From the graph in **Fig. 12**, we can conclude that for 10 flows and a filtation threshold of 1, the maximal length of the the sequences searched maximising the performance of the VDV algorithm is 4 bits, as we favor the large DLL traces.

The precision curves of the VDV algorithm parameterised by the number of flows generated within it will not be presented as this parameter seems to have no influence on this metric.

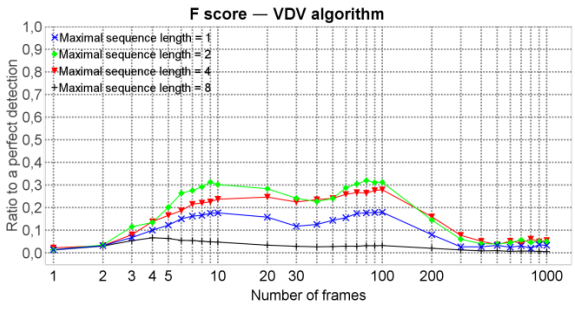

Fig. 12. F score of the VDV algorithm relative to the trace size and parameterized by maximal sequence length

The graph in **Fig. 13** can be split into three parts along the axis of the number of frames of the DLL trace. Below 20 frames, the recall of the VDV algorithm is inversely proportional to the number of frames. This is due to the fact that the variance calculations performed within each flow lose their meaning if the flows contain too few frames, resulting in better coverage for a lower number of flows.

From 20 to 100 frames, the performances are approximately equivalent, as this is an intermediate area between small and large traces.

From 100 to 1000 frames, the recall is proportional to the number of flows, because by increasing the diversity of the flows, full advantage is taken of the two-level variance calculation performed.

We also notice a behaviour relative to the trace size identical to that of the previous curves in **Fig. 10** and **Fig. 11**, which can be explained by the same reasons.

The F score curves being no more interesting than the precision curves, we can say that for a maximal length of the sequences searched of 8 and a filtation threshold of 1, the best number of flows is 30 or more, as we favor the large DLL traces.

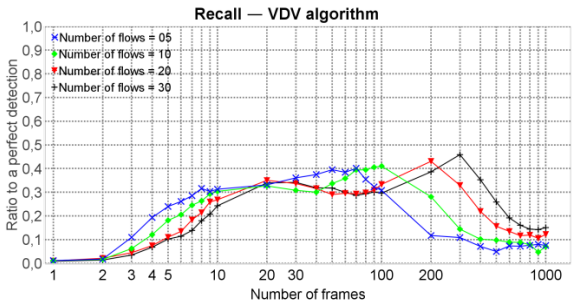

Fig. 13. Recall of the VDV algorithm relative to the trace size and parameterized by the number of flows

The graph in **Fig. 14** shows that the precision of the VDV algorithm is inversely proportional to its filtration threshold. This is due to the fact that as the filtration threshold decreases, only those sequences with the lowest variance of the variances are retained, and this is usually a detectable sequence.

The same behaviour with respect to the number of frames is observed, similar to all other performance curves of the VDV algorithm, but it can be seen that the value of the filtration threshold is indeed the parameter influencing the critical size of the DLL trace above which performance collapses. It would probably have been more appropriate to tie this threshold to the number of frames in the trace.

The identical precision regardless of the filtration threshold for a trace of less than 10 frames is related to the fact that the variance is so big that a large proportion of the tokens are detected, and therefore there are a lot of false positives.

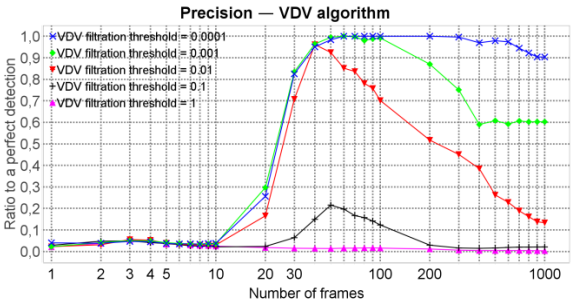

Fig. 14. Precision of the VDV algorithm relative to the trace size and parameterized by the filtration threshold value

We can observe on the graph in **Fig. 15** the collapse in performance when the filtration threshold is too high, characteristic of our implementation of VDV. However, we can also confirm that a threshold independent of the trace size is not at all relevant; indeed, before 7 frames, all curves display the same rapid growth, but afterwards, we can see that each of them seems to have approximately the same behaviour: a phase of decrease, then growth, and finally decrease, but shifted on the axis of the number of frames of the trace. We can therefore assume that for each trace size there exists a corresponding optimal filtration threshold.

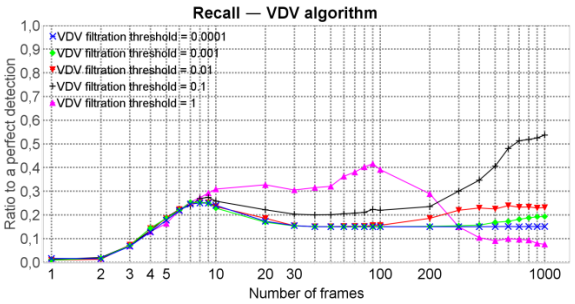

Fig. 15. Recall of the VDV algorithm relative to the trace size and parameterized by the filtration threshold value

From the graph in **Fig. 16**, we can conclude that for a maximal length of the sequences searched of 8 and 10 flows, the filtration threshold maximising the performance of the VDV algorithm over the simulated interval is 0.001, which is only just beginning its growth phase. However, if we were to consider a larger trace, 0.0001 would probably be a better alternative, although ideally the filtration threshold should be indexed to the size of the DLL trace.

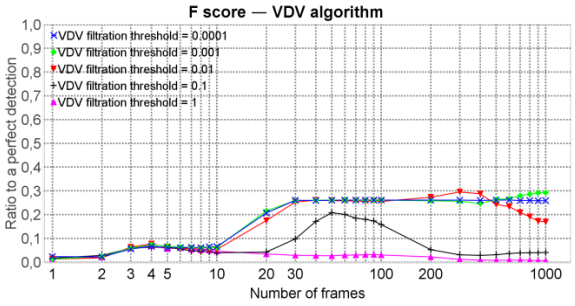

Fig. 16. F score of the VDV algorithm relative to the trace size and parameterized by the filtration threshold value

3.4. Study of AC parameters influence

We then run our implementation of the AC algorithm with a series of parameter sets to study their influence.

Table 5 summarizes the parameters used for the simulations of the AC algorithm.

Table 5. Summary of the AC algorithm simulation parameters

Parameters	Sequences length influence	Filtration threshold influence	Fusion threshold influence
Number of frames	1 to 1000, logarithmic increment	1 to 1000, logarithmic increment	1 to 1000, logarithmic increment
Sequences length	1 to 8, power of 2 increment	8	8
Filtration threshold	1	0.7 to 1.5, increment of 0.1	1
Fusion threshold	1	1	0.1 to 1, increment of 0.1

It can be seen from the graph in **Fig. 17** that the precision of the AC algorithm is inversely proportional to the maximal length of the sequences

searched, similar to the VDV algorithm, and for the same reasons.

The missing points for small trace sizes of the curve corresponding to a maximal sequence length of 1 bit means that no sequences were detected, which is not surprising as applying statistics to very short sequences on a small number of frames does not tend to lead to any tendencies, so filtration may result in nothing being selected.

It can also be seen that the precision increases logarithmically with the size of the DLL trace, but unlike VDV, the precision does not eventually collapse when the number of frames increases. This can be explained by comparing the formulas for computing the filtration thresholds of the two algorithms: VDV's filtration threshold does not depend on the trace size, whereas AC's does. As we previously assumed, such a threshold is more relevant.

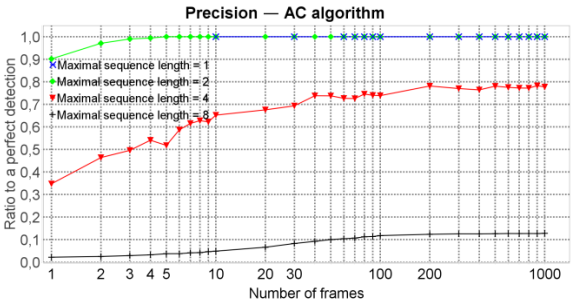

Fig. 17. Precision of the AC algorithm relative to the trace size and parameterized by maximal sequence length

In the graph **Fig. 18**, we can see that the recall of the AC algorithm is proportional to the maximal length of the sequences searched, similar to the VDV algorithm, and for the same reasons.

We also observe the same logarithmic growth similar to the previous curves in **Fig. 17**, and for the same reasons.

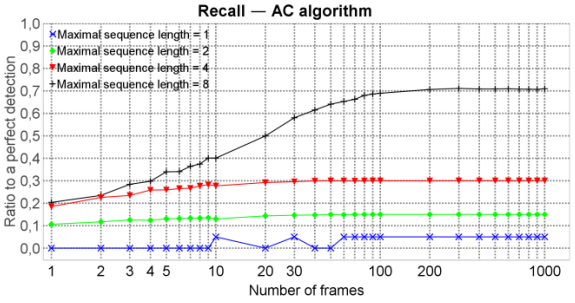

Fig. 18. Recall of the AC algorithm relative to the trace size and parameterized by maximal sequence length

From the graph in **Fig. 19**, we can conclude that for a filtration and fusion threshold of 1, the maximal length of the sequences searched maximising the performance of the AC algorithm is 4 bits.

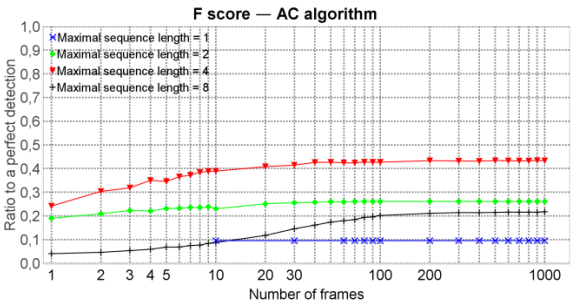

Fig. 19. F score of the AC algorithm relative to the trace size and parameterized by maximal sequence length

The graph in **Fig. 20** shows us that the precision of the AC algorithm increases with the filtration threshold at first, then, beyond 1.3, it decreases. The increase of the filtration threshold means that the filtering is more restrictive, so that, initially, the previously retained sequences that are no longer retained are mainly false positives, hence the improvement in precision. However, after a certain stage, the filtering becomes too hard, and removes more true positives than false positives, which explains the presence of this extremum.

The same logarithmic growth is also observed, similar to all AC performance curves, and for the same reasons.

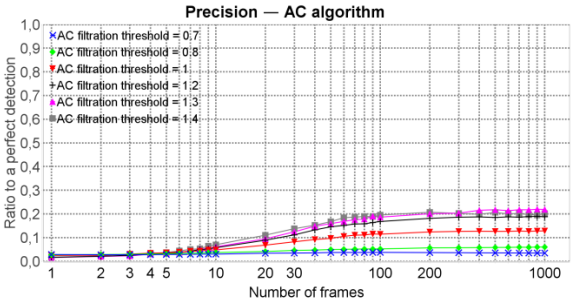

Fig. 20. Precision of the AC algorithm relative to the trace size and parameterized by the filtration threshold value

It can be seen on the graph in **Fig. 21** that the recall of the AC algorithm is inversely proportional to the value of the filtration threshold, but only from a value of 0.8, because the recall stops improving below this value. This simply means that all detectable sequences with a length less than or equal to 8 bits have been detected. The missing 15% is caused by the 3 16-bit sequences present in the DLL trace.

From the graph in **Fig. 22**, we can conclude that

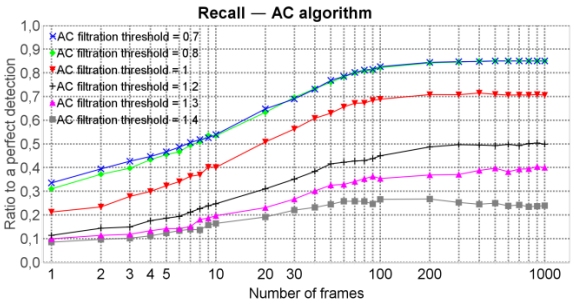

Fig. 21. Recall of the AC algorithm relative to the trace size and parameterized by the filtration threshold value

for a maximal length of the sequences searched of 8 and a fusion threshold of 1, the filtration threshold value maximising the performance of the AC algorithm is 1.2 or 1.3.

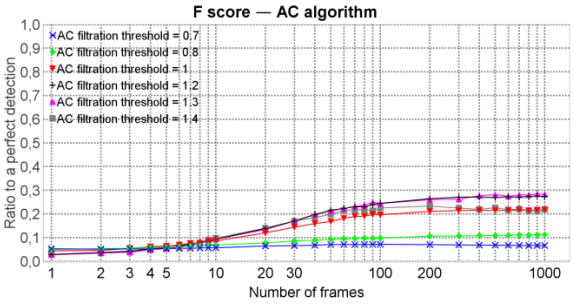

Fig. 22. Fscore of the AC algorithm relative to the trace size and parameterized by the filtration threshold value

The graph in **Fig. 23** shows us that the precision of the AC algorithm is inversely proportional to the fusion threshold between 0.2 and 0.9, and does not vary outside these limits. A low fusion threshold means that relatively different sequences are merged, so that the total number of detected sequences decreases. It appears, according to the graph, that the number of false positives decreases faster than the number of true positives, which means that false positives are assimilated with true positives, causing

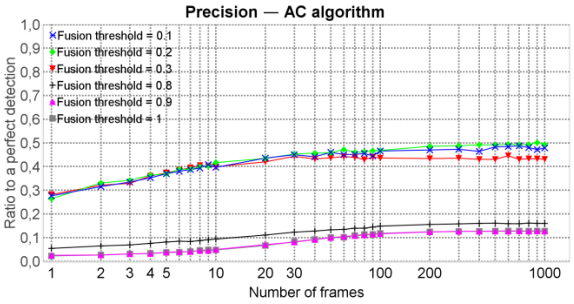

Fig. 23. Precision of the AC algorithm relative to the trace size and parameterized by the fusion threshold value

an increase in precision. However, when the threshold reaches 0.2, for each sequence length, all the sequences are merged into one, making it impossible to gain anything more by merging. On the contrary, when the threshold reaches 0.9, it becomes impossible to merge sequences, hence a non-existent impact of the merging procedure.

We can see in the graph **Fig. 24** that the recall of the AC algorithm is proportional to the fusion threshold between 0.2 and 0.9, and does not vary outside these boundaries. This seems logical, because a higher fusion threshold implies less sequence merging, and therefore greater diversity, leading to better recall. The visible boundaries are due to the same reasons as for the graph in **Fig. 23**.

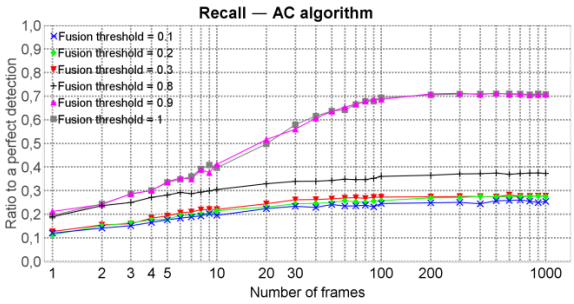

Fig. 24. Recall of the AC algorithm relative to the trace size and parameterized by the fusion threshold value

From the graph **Fig. 25**, we can conclude that for a maximal length of 8 of the sequences searched and a filtration threshold of 1, the fusion threshold of the sequences maximising the performance of the AC algorithm is 0.2.

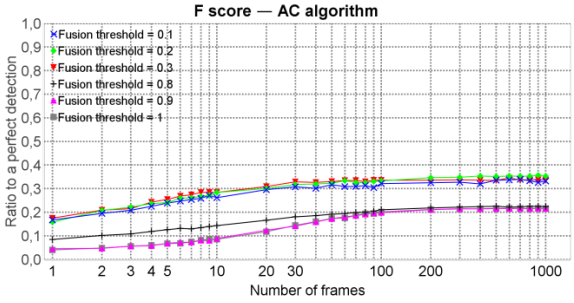

Fig. 25. Precision of the AC algorithm relative to the trace size and parameterized by the fusion threshold value

3.5. Study of LDA parameters influence

We finally run our implementation of the LDA algorithm with a series of parameter sets to study their influence.

We do not present the performance curves parameterized by Beta because this parameter has

virtually no influence given the parameter values we simulated. Beta influences the a priori on the concentration of the multinomial distribution of the words over topics. We simulated it for values inferior to 1, so it means it just influenced the gap between high and low probability values.

However, with the maximal probability gradient value we used, the selected keywords would always be the ones above the probability gap, whatever the gap width.

Table 6 summarizes the parameters used for the simulations of the LDA algorithm.

Table 6. Summary of the LDA algorithm simulation parameters

Parameters	**Sequences length influence**	**Keywords number influence**	**Maximal perplexity gradient influence**	**Maximal probability gradient influence**	**Alpha influence**
Number of frames	1 to 1000, logarithmic increment	1 to 1000, logarithmic increment	1 to 1000, logarithmic increment	1 to 1000, logarithmic increment	1 to 1000, logarithmic increment
Sequences length	1 to 8, power of 2 increment	8	8	8	8
Keywords number	10	3 5 10 20 30	10	10	10
Maximal perplexity gradient	1	1	0.1 to 10, logarithmic increment	1	1
Maximal probability gradient	0.1	0.1	0.1	0.01 to 1, logarithmic increment	0.1
Alpha	1	1	1	1	0.01 to 100, power of 10 increment
Beta	0.0001	0.0001	0.0001	0.0001	0.0001

It can be seen on the graph in **Fig. 26** that the precision of the LDA algorithm is inversely proportional to the maximal length of the sequences searched, similar to the VDV and AC algorithms, and for the same reasons.

There is also an approximately linear growth proportional to the size of the DLL trace, indicating that, unlike the AC algorithm, a learning-based technique keeps increasing its performance as its learning base becomes larger, making it more suitable for large volumes of data.

In the graph **Fig. 27**, we can see that the recall of the LDA algorithm is proportional to the maximal length of the sequences searched, similar to the VDV and AC algorithms, and for the same reasons.

We notice a phase of slight increase, then slight decrease of the recall, as the size of the DLL trace grows. This can be explained by the search for too few keywords, which causes most of the keywords to converge towards a limited number of sequences as the number of frames increases, resulting in a reduction in recall. However, when the maximal length of the sequences searched becomes long enough, this phenomenon seems to disappear.

From the graph in **Fig. 28**, we can conclude that for 10 keywords, a maximal perplexity gradient of 1,

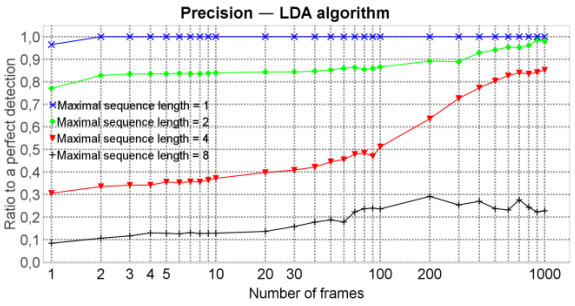

Fig. 26. Precision of the LDA algorithm relative to the trace size and parameterized by maximal sequence length

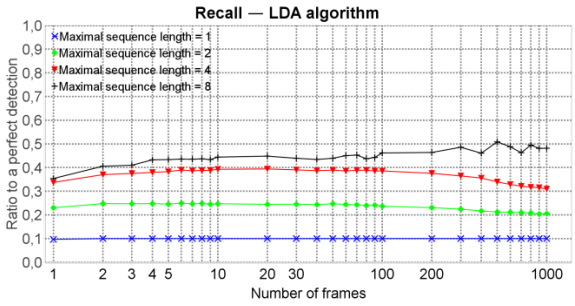

Fig. 27. Recall of the LDA algorithm relative to the trace size and parameterized by maximal sequence length

a maximal probability gradient of 0.1, an alpha of 1, and a beta of 0.0001, the maximal length of the sequences searched maximising the performance of the LDA algorithm is 4 bits.

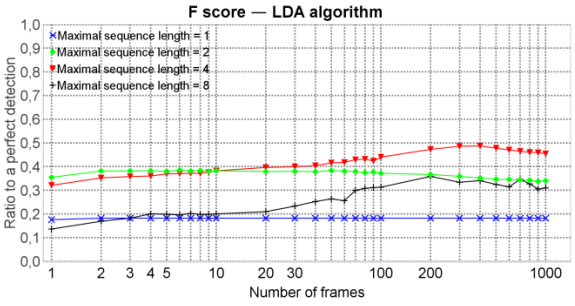

Fig. 28. F score of the LDA algorithm relative to the trace size and parameterized by maximal sequence length

We can observe on the graph in **Fig. 29** that the precision of the LDA algorithm is inversely proportional to the number of keywords assumed to be present in the DLL trace. Indeed, since only the most probable sequences are selected for each keyword, if fewer keywords are assumed, then only the most probable sequences among those will be selected. These most probable sequences are generally remarkable sequences, which means more true positives, and hence higher precision.

In addition, there are generally two phases: a linear growth with respect to the size of the DLL trace, for the same reason as for the previous LDA curves in **Fig. 26-28**; and a constant phase corresponding to the maximal performance achievable with the value given to the other parameters.

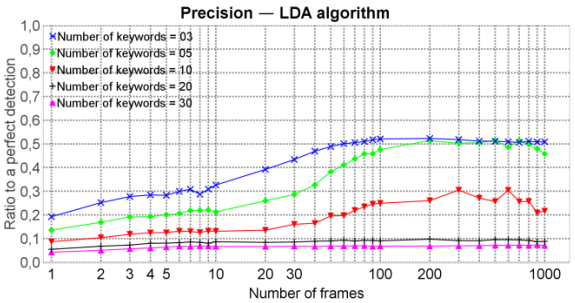

Fig. 29. Precision of the LDA algorithm relative to the trace size and parameterized by the number of keywords

In the graph **Fig. 30**, we can see that the recall of the LDA algorithm is proportional to the number of keywords assumed in the DLL trace. This seems logical, because the greater the number of keywords, the greater the number of sequences selected, and therefore the greater the recall.

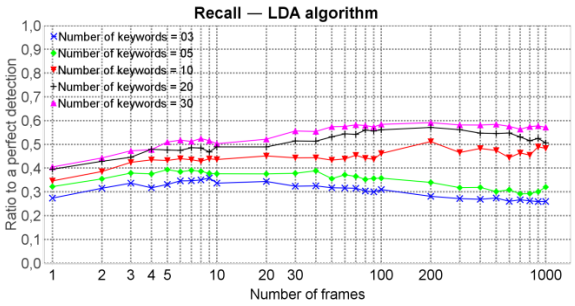

Fig. 30. Recall of the LDA algorithm relative to the trace size and parameterized by the number of keywords

From the graph in **Fig. 31**, we can conclude that for a maximal length of 8 of the sequences searched, a maximal perplexity gradient of 1, a maximal probability gradient of 0.1, an alpha of 1, and a beta of 0.0001, the number of keywords maximising the performance of the LDA algorithm over the simulated interval is 3 or 5, but we will favour larger DLL trace sizes, so we will select 5.

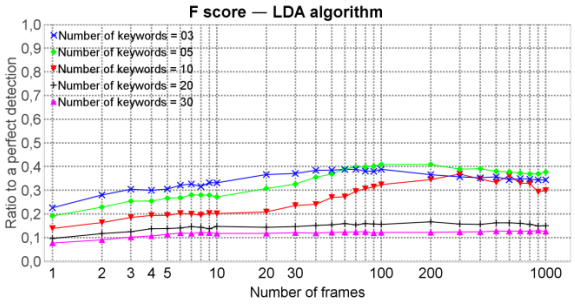

Fig. 31. F score of the LDA algorithm relative to the trace size and parameterized by the number of keywords

We can see on the graph in **Fig. 32** that the precision of the LDA algorithm is proportional to the value of the maximal perplexity gradient before the Gibbs sampler stops, between 0.3 and 3, and then

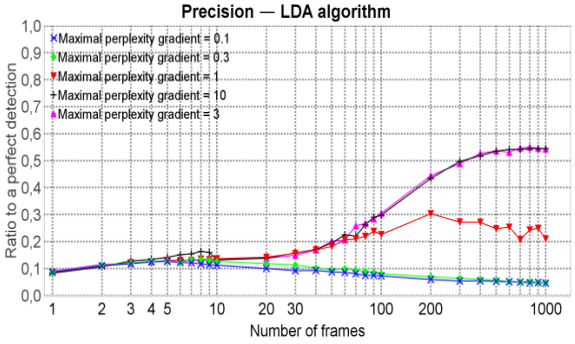

Fig. 32. Precision of the LDA algorithm relative to the trace size and parameterized by maximal perplexity gradient

stabilises. This means that iterating the sampler a large number of times is useless, and even counter-productive, which is very surprising.

The graph in **Fig. 33** shows that the recall of the LDA algorithm is inversely proportional to the maximal perplexity gradient. This means that iterating the Gibbs sampler a greater number of times favours a greater diversity of keywords, and therefore of sequences selected for these keywords.

The behaviour of the reacall relative to the size of the DLL trace is similar to the previous recall curves of the LDA algorithm in **Fig 27** and **Fig. 30**, and for the same reason.

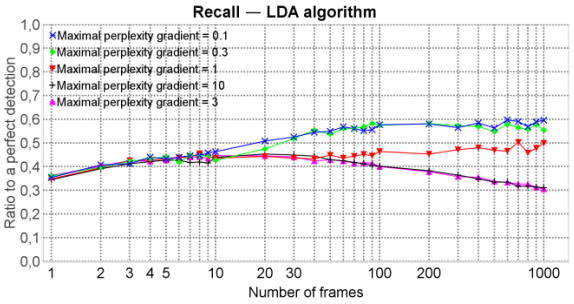

Fig. 33. Recall of the LDA algorithm relative to the trace size and parameterized by maximal perplexity gradient

From the graph in **Fig. 34**, we can conclude that for 10 keywords, a maximal length of 8 of the sequences searched, a maximal probability gradient of 0.1, an alpha of 1, and a beta of 0.0001, the values of the maximal perplexity gradient maximising the performance of the LDA algorithm are those greater than or equal to 3.

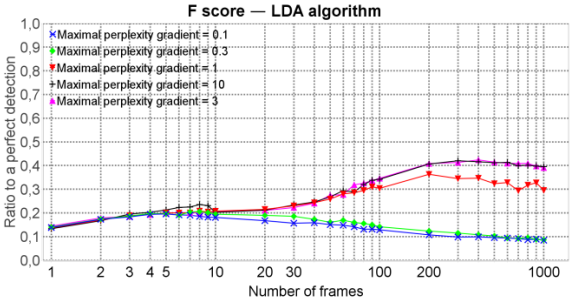

Fig. 34. F score of the LDA algorithm relative to the trace size and parameterized by maximal perplexity gradient

We can see on the graph in **Fig. 35** that the precision of the LDA algorithm is inversely proportional to the maximal probability gradient. This seems logical, as the lower this gradient is, the more the n-grams with the highest probability within each keyword will be favored, thus those most likely to be remarkable sequences of the protocol.

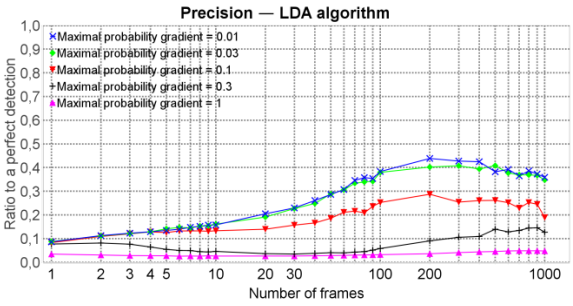

Fig. 35. Precision of the LDA algorithm relative to the trace size and parameterized by maximal probability gradient

We also notice a behaviour generally resembling that of the precision parameterized by the number of keywords, with a growth phase, then a stabilization. The ceiling is probably due to the too high number of keywords and the too low perplexity gradient.

The graph in **Fig. 36** shows that the recall of the LDA algorithm is proportional to the maximal probability gradient. This is due to the fact that as this gradient increases, the hardness of the n-gram selection decreases, therefore more n-grams are detected, resulting in an increase in diversity, and therefore recall.

The behaviour with respect to the number of frames is similar to that of all other recall curves of the LDA algorithm, and for the same reasons.

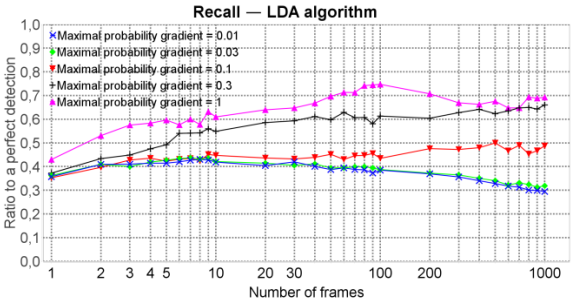

Fig. 36. Recall of the LDA algorithm relative to the trace size and parameterized by maximal probability gradient

From the graph in **Fig. 37**, we can conclude that for 10 keywords, a maximal length of the sequences searched of 8, a maximal perplexity gradient of 1, an alpha of 1, and a beta of 0.0001, the value of the maximal probability gradient maximising the performance of the LDA algorithm is 0.03 or less.

On the graph in **Fig. 38**, we can see that the precision of the LDA algorithm is proportional to Alpha. By studying the characteristics of Dirichlet law, we can say that this means that the algorithm offers its best precision when we consider that the frames are composed of a relatively homogeneous mixture of all the keywords of the protocol, rather

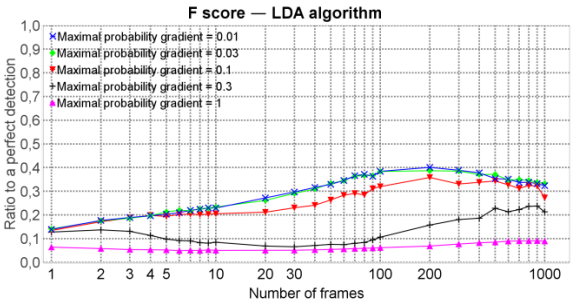

Fig. 37. F score of the LDA algorithm relative to the trace size and parameterized by maximal probability gradient

than just a few. This seems to correspond relatively well to reality, so these results are not surprising.

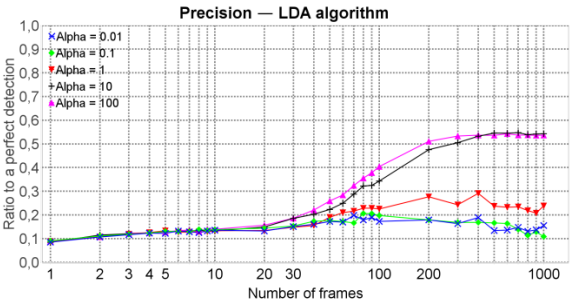

Fig. 38. Precision of the LDA algorithm relative to the trace size and parameterized by alpha

The graph in **Fig. 39** shows that the recall is inversely proportional to Alpha. Although we do not have a precise explanation for this behaviour, it is consistent with the principle verified in all the previous curves in **Fig. 10-38**, namely that precision and recall always show opposite trends when varying a parameter. Furthermore, the improvement in accuracy as Alpha increases is greater than the deterioration of the recall, resulting in an overall improvement in performance, which is in line with our hypothesis when observing the graph in **Fig. 38**.

From the graph **Fig. 40**, we can conclude that for

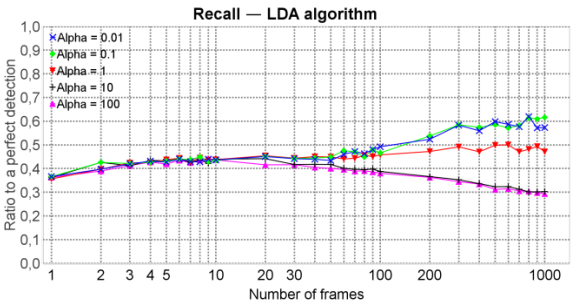

Fig. 39. Recall of the LDA algorithm relative to the trace size and parameterized by alpha

10 keywords, a maximal length of 8 of the sequences searched, a maximal perplexity gradient of 1, a maximal probability gradient of 0.1, and a beta of 0.0001, the values of Alpha maximising the performance of the LDA algorithm are those greater than or equal to 10.

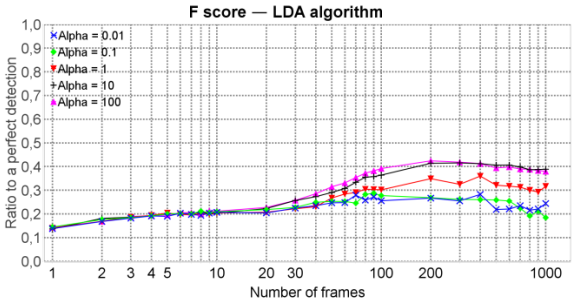

Fig. 40. F score of the LDA algorithm relative to the trace size and parameterized by alpha

3.6. Comparison of the best parameter sets

For each graph, we selected the best performing curve in the previous subsection and retrieved its corresponding algorithm parameters. Then, out of all these best performing curves, we select the best one for each algorithm, and address them in this subsection. We obtain the following parameter sets :

- For all algorithms, a maximal length of the sequences searched of 4
- For the VDV algorithm, a number of flows randomly generated of 10 and a filtration threshold coefficient of 1.
- For the AC algorithm, a fusion threshold and a filtration threshold of 1.
- For the LDA algorithm, a number of keywords of 10, a maximal perplexity gradient of 1, a maximal probability gradient of 0.1, an alpha of 1, and a beta of 0.0001.

The graph in **Fig. 41** shows that, for small DLL traces (less than 400 frames), the AC algorithm offers the best precision, followed by the LDA algorithm, and finally the VDV algorithm. For traces of more

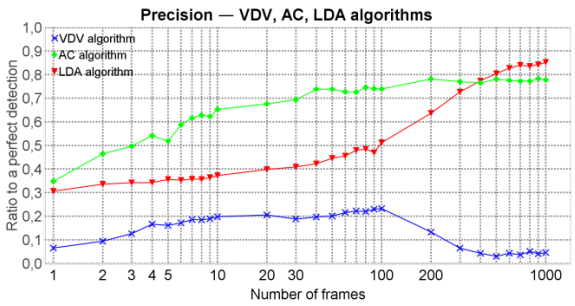

Fig. 41. Best precision of the algorithms VDV, AC, and LDA

than 400 frames, the LDA algorithm becomes more precise than the AC algorithm, and the improvement in precision seems to continue beyond 1000 frames.

Maximum performance is around 85% precision for LDA, and just under 80% for AC, while VDV peaks at just over 20%.

On the graph in **Fig. 42**, we can see that the LDA algorithm has the best recall regardless of the size of the trace. Between 10 and 100 frames, the VDV algorithm has the second best recall, and the AC algorithm has the worst one, but outside these limits, AC is second, and VDV last.

Maximum performance is about 40% recall for LDA, 35% for VDV, and 30% for AC.

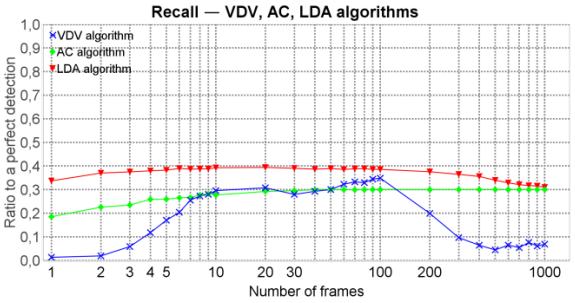

Fig. 42. Best recall of the algorithms VDV, AC, and LDA

The graph in **Fig. 43** shows that the AC and LDA algorithms have approximately identical performance, with the LDA looking slightly better. The VDV algorithm offers significantly poorer results, which even determining the filtration threshold based on the size of the DLL trace could not fully compensate for; at most, results at 1000 frames would be on the same order as of those obtained for 100.

Maximum performance is slightly under 50% F score for LDA, slightly under 45% for AC, and slightly under 30% for VDV.

Let us now switch from the quality of the sequence detection to its usefulness for field detection.

The graph in **Fig. 44** shows that the average fields lengths detection ratio of the LDA algorithm is the best, followed by that of AC, and finally VDV.

Maximum performance is slightly less than 60% for LDA, a bit under 50% for AC, and slightly under 30% for VDV.

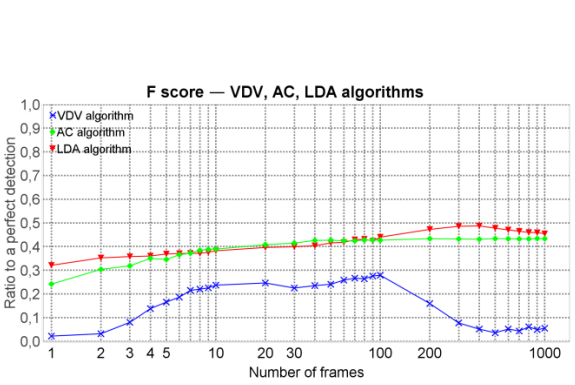

Fig. 43. Best F score of the algorithms VDV, AC, and LDA

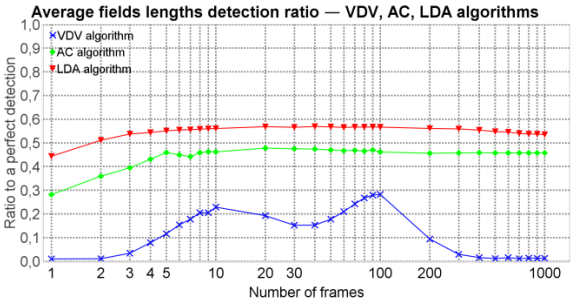

Fig. 44. Best average fields lengths detection ratio of the algorithms VDV, AC, and LDA

The same behaviour can be seen on the graphs in **Fig. 45** and **Fig. 46**, concerning the average fields values and fields positions detection ratios. Their maximum performances are, respectively, 75% for the LDA algorithm, slightly above 40% for AC, slightly below 40% for the VDV algorithm, and slightly below 60% for the LDA algorithm, slightly

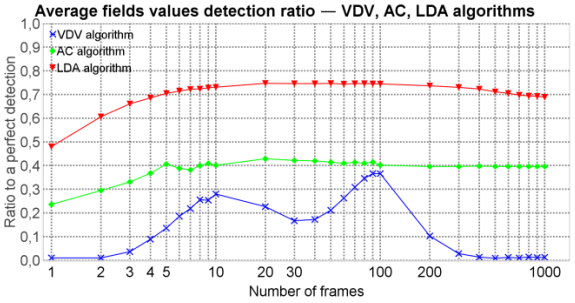

Fig. 45. Best average fields values detection ratio of the algorithms VDV, AC, and LDA

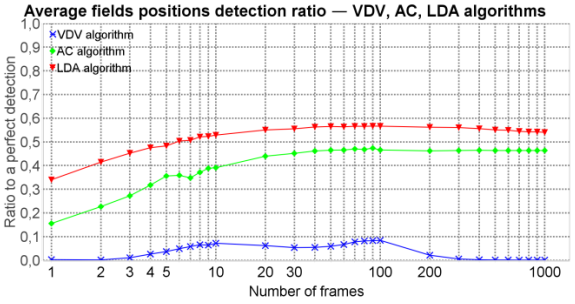

Fig. 46. Best average fields positions detection ratio of the algorithms VDV, AC, and LDA

below 50% for AC, and slightly below 10% for the VDV algorithm.

We summarize the relative performance of the three algorithms in **Table 7**. The number of stars stands, by decreasing order, for the best, average, and worst.

Table 7. Relative performance summary of the algorithms VDV, AC, and LDA

Algorithms	**VDV**	**AC**	**LDA**
Precision	*	***	***
Recall	*	**	***
F score	*	***	***
Average fields lengths detection ratio	*	**	***
Average fields values detection ratio	*	**	***
Average fields positions detection ratio	*	**	***
Total	6	14	18

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 by analysing larger traces, 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, and design our own Bayesian network model.

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]. P. -S. Gréau-Hamard, M. Djoko-Kouam, Y. Louet, A comparative Study of Sequence Identification Algorithms in IoT Context, in *2020 Proceedings of the 2nd International Conference on Advances in Signal Processing and Artificial Intelligence (ASPAI' 2020)*, November 2020, pp. 137-143.

[9]. 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.

[10]. 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.

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

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

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

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

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