Microelectronics Journal 43 (2012) 77–87

Contents lists available at SciVerse ScienceDirect

Microelectronics Journal

journal homepage: www.elsevier.com/locate/mejo

Two effective methods to detect anomalies in embedded systems

Mahroo Zandrahimi, Hamid R. Zarandi *, Mohammad H. Mottaghi

Department of Computer Engineering and Information Technology, Amirkabir University of Technology (Tehran Polytechnic), Iran

ARTICLE INFO

Article history:
Received 1 January 2011
Received in revised form
29 October 2011
Accepted 8 November 2011
Available online 25 November 2011

Keywords:
Anomaly detection
Fault detection
Embedded systems
Fault coverage
Sensor data

ABSTRACT

Current-day embedded systems are very vulnerable to faults and defects. Anomaly detection is often the primary means of providing early indication of faults and defects. This paper presents two methods for detecting anomalies in embedded systems. The first method, buffer based detector, constructs a buffer consisting of events from a stream of data considered to be normal. Consequently, during test stage, if an event does not exist in the buffer, a miss will happen. An anomaly exists in test data provided that the hit rate of the buffer does not reach a predefined threshold. The second method namely probabilistic detector employs the probability of data events to evaluate the behavior of system. In order to measure the probability of events in the system, sampling of two events with distinct distance is done. Eventually, during test stage, the probability of events can be measured. An anomaly exists in test data provided that this probability does not reach a predefined threshold. A comparison between these two methods and other typical methods has been done based on detection coverage, area overhead and delay overhead. The experiments on 112 standard benchmarks show that the proposed methods can detect 100% of anomalies. Also, the area overhead of the proposed detectors grows linearly, while the area overhead of other typical detectors grows exponentially by the increase in one of the detector's parameters.

1. Introduction

As computer systems become smaller and smaller, they will be embedded in every device, such as modern automotives [1], airplanes, air traffic controllers [2], military applications [3], hospital equipments [4], and nuclear equipments [5]. Many of these devices with embedded computers will be safety critical that any bugs can affect serious damage on human life, and therefore will require a higher level of dependability than usual. It is presumed that fault tolerance, which is one way of achieving high dependability, will be employed in such devices. In several fault tolerance methods, the first step requires that fault be detected and, then, bringing fault tolerance measures to tolerate it. Hence, fault detection is an essential first step in achieving dependability [6].

Anomaly detection refers to the problem of finding patterns in data that do not conform to expected behavior. These non-conforming patterns are often referred to as anomalies, outliers, discordant, observations, exceptions, aberrations, surprises, peculiarities, or contaminants in different application domains. Of these, anomalies and outliers are two terms used most commonly in the context of anomaly detection; sometimes interchangeably [7].

The importance of anomaly detection is due to the fact that anomalies in data translate to significant actionable information in a wide variety of application domains. For example, an anomalous traffic pattern in a computer network could mean that a hacked computer is sending out sensitive data to an unauthorized destination [8]. An anomalous MRI image may indicate presence of malignant tumors [9]. Anomalies in credit card transaction data could indicate credit card or identity theft [10] or anomalous readings from a space craft sensor could signify a fault in some component of the space craft [11].

Faults can be detected either explicitly or implicitly. In the former, faults are typically detected through pattern recognition, which aims to classify data (patterns) based either on a priori knowledge or on statistical information extracted from the patterns [12]. In the latter, fault is detected due to some indirect indicator, such as anomaly, that may have been caused by the fault. In such systems, many different measures are available for unusual behaviors. For example, there can be many sensors measuring the state of a system [13]. These sensors can be hardware or software. The data produced by such sensors are referred to as sensor-data that can be numeric or categorical [6]. Numeric data that are usually continuous are on a ratio scale and have a unique zero point; besides, they have mathematical ordering properties; for example, ten is twice five. Categorical

* Corresponding author.
E-mail addresses: mzand@aut.ac.ir (M. Zandrahimi), h_zarandi@aut.ac.ir (H.R. Zarandi), mh_mottaghi@aut.ac.ir (M.H. Mottaghi).

0026-2692/$ - see front matter © 2011 Elsevier Ltd. All rights reserved.
doi:10.1016/j.mejo.2011.11.003

data are discrete; they usually consist of a series of unique labels as categories and have no mathematical ordering properties; for example, blue cannot be added to red [14]. It seems likely that as computing power increases, more of the sensor data will be in the form of categorical data [7]; hence, anomaly detectors will be required to operate primarily on categorical data, presenting a real challenge to developers and users of such sensors because categorical data are much more difficult to handle statistically than numeric data are. This paper focuses on detecting anomalies in categorical data.

An anomaly occurring in such sensor data is often the indirect or implicit manifestation of a fault or condition somewhere in the monitored system or process. Detecting such anomalies, therefore, can be an important aspect of maintaining the integrity, reliability, safety, security, and general dependability of a system or process. Such anomaly detection is on the front line of many fault-tolerance and dependability mechanisms, it is essential that it is itself, reliable. One way to measure its reliability is by its coverage. Coverage is a figure of merit that measures the effectiveness of a detection or testing process. Historically, a system's coverage has been said to be the proportion of faults from which a system recovers automatically; the faults in this class are said to be covered by recovery strategy. In this paper, coverage is viewed as the probability that a particular class of conditions or events is detected before a system suffers consequences from a missed or false detection.

An anomaly is an event or object that differs from some standard or reference event, in excess of some threshold, in accordance with some similarity or distance metric on the event [6]. The reference event is what characterizes normal behavior. The "similarity metric" measures distance between normal and abnormal data. The threshold establishes the minimum distance that encompasses the variation of normalcy which means that any event exceeding that distance is considered anomalous. In categorical data, anomalous events are typically defined by the probabilities of encountering particular juxtapositions of symbols or subsequences in the data stream; i.e., symbols and sequences in an anomaly are juxtaposed in an unexpected way.

Categorical data sets are comprised of sequences of symbols [15]. The collection of unique symbols in a data set is called the alphabet. Training data are obtained from the process over a period of time during which the process is judged to be running normal. The juxtapositions of symbols and sequences within these data would be considered normal because no faults occurred during the collection period. Once the training data are characterized which is termed the training phase, characterizations of new data, monitored while the process is in an unknown state, are compared to expectations generated by the training data. Any sufficiently unexpected juxtaposition in the new data would be judged anomalous, the possible effect of a fault.

In threshold-based algorithms, the more a calculated value exceeds a *threshold*, the more anomalous the event will seem. It should be noted that each threshold value has direct impact on false alarm. If the threshold value is set low, most of training data may generate false alarms (*False Positive*) and on the other hand, if the threshold value is set high, most of anomalous data may be passed without any alarm (*False Negative*).

Threshold setting in anomaly detection systems can be classified in two cases: (1) fixed threshold setting [26,27], and (2) adaptive threshold setting [28,29]. In the former, one threshold is determined in training phase and used for all future data in testing phase. However, in the latter, threshold is renewed during run time because the input behavior of the systems may be changed dynamically. In the context of embedded systems, fixed thresholds have been used in related works since most of these systems are utilized in some known specified conditions whose inputs' behaviors are pre-defined [6,24]. Therefore, the fixed threshold setting is used in this work like previous works in embedded systems [6,24].

Hence, in this paper threshold setting is performed fixed and based on an acceptable rate of *False Positive*. The threshold value is tuned based on following equation:

$$Threshold = Min\{t \mid fp(S,t) \le \rho\} \quad (1)$$

where t is threshold random variable, S is set of all training data, $fp(S,t)$ is a function of S and t whose value is *False Positive* percent of a given anomaly detection algorithm on set S for given threshold t. Moreover, ρ is acceptable percent of *False Positive*. In our experiments, without loss of generality, we supposed the value of ρ equal to 5%. This equation finds minimum value of all thresholds that generate *False Positive* rate less than 5% for given set of training data. We choose the minimum threshold because the less the threshold value is set, the more detectability of testing data the method has.

This paper presents two methods for detecting anomalies in embedded systems. The first method, buffer-based detector, uses a buffer to store the extracted information from the environment during the period of time that the behavior of the system is judged to be normal. In this method a buffer consisting of events is constructed from the training data stream. Eventually, during test stage, if an event does not exist in the buffer, a miss will happen. An anomaly exists in test data provided that the hit rate of the buffer does not reach a predefined threshold. The second method namely probabilistic detector uses the relative distance between symbols in training data. This method employs the probability of events to evaluate the behavior of system in training stage, and if this probability does not reach a predefined threshold, an anomaly exists in test data. A comparison between these two methods and other typical methods has been done based on detection coverage, area overhead and delay overhead. The experiments on 112 standard benchmarks show that the proposed methods can detect 100% of anomalies. Also, the area overhead of the proposed detectors grows linearly, while the area overhead of other typical detectors grows exponentially by the increase in the size of detector window.

The remainder of this paper is organized as follows: previous works related to this paper are studied in Section 2, formulating the problem is presented in Section 3; the proposed methods are introduced in Section 4. An accurate estimation of the number of memory elements required for each detector is calculated in Section 5. Section 6 is devoted to the procedure of generating the standard benchmark data sets, experimental results are given in Section 7, and finally Section 8 concludes the paper.

2. Previous works

Anomaly detection is an important problem that has been researched within diverse research areas and application domains. A lot of works have been done in network era to deal with anomalies [16–23]. As a quick review, in [16], a method based on Bayesian belief networks is proposed. This method captures the conditional dependencies among the observations of the attributes to detect the anomalies in the sensor streamed data. In [17], the authors addressed the problem of unsupervised anomaly detection in wireless sensor networks. The proposed algorithm works based on communication with all nodes to reveal an abnormal behavior. In [19], the authors proposed a general scheme to detect anomalies that are caused by adversaries. The advantage of this method is that it is independent of the network topology; however, this method can detect localized anomalies only. None of these mentioned methods in [16–23]

have been specially proposed for anomaly detection in embedded system, which is the focus of this paper.

Two typical methods that have been presented specifically for detecting anomalies in embedded systems are Markov and Stide anomaly detectors [6,24]. The Markov anomaly detector determines whether the states in the sequential data stream, taken from a monitored process, are normal or anomalous. It calculates the probabilities of the transitions between events in a training set, and uses these probabilities to assess the transition between events in a testing set. These states and probabilities can be described by a Markov model. The key aspect of Markov model is that the future state of the modeled process depends only on the current state, and not on any previous state. A Markov model consists of a collection of all possible states and a set of probabilities associated with transitioning from one state to another. Basing an anomaly detector on a discrete Markov process requires three steps. First, a state transition matrix is constructed, using the training data; the training data represent the conditions that are considered to be normal. The second step is to establish a threshold that determines how dissimilar from normal a process can be, while remaining within the bounds of what is considered to be normal operating behavior. The third step is to examine the test data if they fall within the expectations established by the training data. Although the coverage of this detector is relatively high, it imposes a great deal of area overhead which grows exponentially by the increase in the size of detector window.

The other method, Stide is a sequence, time-delay, embedding anomaly detector inspired by natural immune systems that distinguish self (normal) from nonself (anomalous). The reference to "time" recognizes the time-series nature of categorical data on which the detector is typically deployed. Stide mimics natural immune systems by constructing templates of "self" and, then, matching them against instances of "nonself". Stide anomaly detector requires three steps. First, a database consisting of templates of "self" is constructed from a stream of data considered to be normal. The second step is to compare sequences from an unknown dataset to each of its "self" sequences. Any unknown sequence that does not match a "self" sequence is termed a mismatch. Finally, a score is calculated on the basis of the number of sequence comparisons made within a temporally localized region. This detector is merely able to detect anomalies of certain classes; besides, the area overhead imposed by this detector is relatively high.

3. Formulating the problem

This section presents the terminology and the definitions related to the proposed method. The following definitions are inspired from [6].

Fault could manifest itself as an event injected into a normal stream of data, and that event could be regarded as normal or as anomalous. There are three phenomena that could make an event anomalous: (a) Foreign symbols, a foreign symbol is a symbol not included in the training-set alphabet. For example, any symbol, such as a *Q*, not in the training-set alphabet comprising *A B C D E F* would be considered a foreign symbol. Events containing foreign symbols are called foreign symbol-sequence anomalies. (b) Foreign *n*-grams/sequences, an *n*-gram (a set of *n* ordered elements) not found in the training data set (and also not containing a foreign symbol) is considered a foreign *n*-gram or foreign sequence. A foreign *n*-gram event contains *n*-grams not present in the training data. For example, given an alphabet of *A B C D E F*, the set of all bigrams would contain *AA AB AC … FF*, for a total $6^2=36$. If the training data contained all bigrams except *CC*, then *CC* would be a foreign *n*-gram. (c) Rare *n*-grams/ sequences, a rare *n*-gram event, also called a rare sequence, contains *n*-grams that are infrequent in the training data. In the example above, if the bigram *AA* constituted 96% of the bigrams in the sequence, and the bigram *BB* and *CC* constituted 2% of each, then *BB* and *CC* would be rare bigrams.

An anomaly detector determines the similarity or distance between some standard event and the possibly anomalous events in its purview. The purview of a sliding window detector is the length of the window. The extent which the detector window overlaps the anomaly can be thought of as the detector's view of anomaly. Fig. 1 depicts different views of an anomaly injected into a sensor data stream from the perspective of a sliding-window anomaly detector. In this figure anomaly is depicted by *A* symbol, and sensor-data stream is depicted by *d* symbol. The right directed arrows indicate the direction of data flow. The case in which the window is the same size as the anomaly and the entire anomaly is captured exactly within the window is called the *whole view*. When the size of the detector window is less than the length of the anomaly, the detector has what is called an *internal view*. For the case in which the detector window is larger than the anomaly and both anomalous and normal background data are seen, this is the *encompassing view*. In a *boundary condition*, the detector sees part of the anomaly and part of the background data. Boundary conditions that arise at both ends of an injected sequence embedded in normal data occur independently of the relative sizes of the detector window and anomaly. The *background view* sees only background data and no anomalies. These conditions except boundary condition depend on the size of the injected event relative to the size of the detector window.

It is worth mentioning that anomalies can be composed of subsequences of various types, three of which were already identified in this section of revised paper: foreign symbols, foreign *n*-grams, and rare *n*-grams. Here, the terms *n*-gram and sequence will be used interchangeably, i.e., foreign *n*-gram and foreign sequence refer to the same thing.

That an anomalous sequence can be composed of several different kinds of subsequences, along with the concept of internal sequences and boundary sequences, gives rise to the idea of creating a map of the anomaly space for sliding-window detectors. By giving such a map, it should be possible to determine the extent to which that map is covered by a particular anomaly detector. An anomaly-space map is shown in Fig. 2. In this figure the window size of the detector, relative to the size

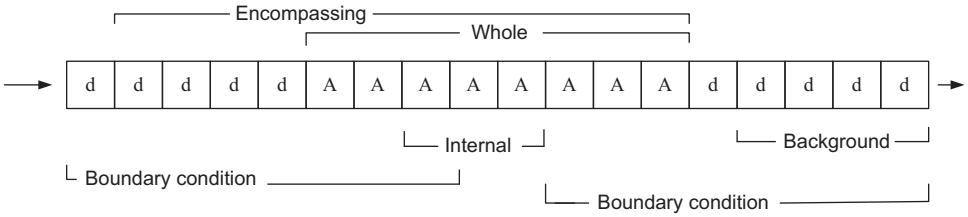

Fig. 1. Different views of an anomaly detector.

Foreign-Symbol-Sequence Anomalies

DW < AS	DW=AS	DW>AS
FF AI AB	FF-AB	FF AE AB

Foreign-Sequence Anomalies

DW<AS	DW=AS	DW > AS
FS AI AB		FS AE AB
FS RI AB	FS--AB	~~FS RE AB~~
FS CI AB		~~FS CE AB~~
FS AI RB		FS AE RB
FS RI RB	FS--RB	~~FS RE RB~~
FS CI RB		~~FS CE RB~~
FS AI CB		FS AE CB
FS RI CB	FS--CB	~~FS RE CB~~
FS CI CB		~~FS CE CB~~

Rare-Sequence Anomalies

DW<AS	DW=AS	DW > AS
RS AI AB		RS AE AB
RS RI AB	RS--AB	~~RS RE AB~~
RS CI AB		~~RS CE AB~~
RS AI RB		RS AE RB
RS RI RB	RS--RB	~~RS RE RB~~
RS CI RB		~~RS CE RB~~
RS AI CB		RS AE CB
RS RI CB	RS--CB	~~RS RE CB~~
RS CI CB		~~RS CE CB~~

Fig. 2. Anomaly space map [6].

of the anomaly, is shown in the three columns of the figure: detector window size less than anomaly size (DW < AS), detector window size equal to the anomaly size (DW = AS), and detector window size greater than the anomaly size (DW > AS). For each of these conditions, the figure addresses three kinds of anomalies: foreign-symbol sequence anomalies (sequences comprising only foreign symbols), foreign-sequence anomalies (sequences comprising only foreign sequences), and rare-sequence anomalies (sequences comprising only rare sequences).

In this figure, the first two letters in each row of each cell identify the type of anomalous sequence; in other words, FS refers to foreign-sequence anomaly, RS refers to rare-sequence anomaly, and FF refers to foreign-symbol-sequence anomaly. The next two letters identify the type of conditions (comprising "I" for internal view and "E" for encompassing view), each of which can be alien (A), rare (R), or common (C). The last two letters refer to the boundary conditions (B), which can also be alien, rare, or common. For the case that the detector window size is equal to the anomaly size, no internal or encompassing conditions occur. These impossible conditions are struck out by dashes replacing the middle two letters.

The following material expands the description of a cell by selecting as an example the anomaly type FF AI AB, depicted at the upper left of the Fig. 2. This is a sequence of foreign symbol (FF) composed of alien internal sequences (AI) and having alien external boundaries (AB). The term *alien* is used to refer to sequences that do not exist in the normal training data, irrespective of the characteristics that make them foreign, unlike the more closely-defined terms *foreign symbol* and *foreign sequence*.

FF is a foreign-symbol-sequence anomaly composed only of foreign symbols. In this specific case, when the anomaly sequence FF AI AB slides past a detector window whose size is less than the size of the anomaly, the detector will first confront the leading edge of the anomaly. The leading edge will be alien, i.e., the sequence containing the first element of the anomaly and the normal element immediately preceding it is not a sequence that exists in the normal training data and, therefore, will be anomalous. As the anomaly moves through the detector window, each internal, detector-window-sized subsequence of the anomaly will be alien. As the anomaly passes out the window, its trailing edge will be form another alien boundary.

4. Proposed methods

4.1. Buffer-based anomaly detector

One method to detect anomaly is through storing the normal behavior of the system in the form of a buffer. In this method, it is tried to store the extracted information from the environment in a buffer during the training stage. Ultimately, by referring to the obtained information in the buffer, the behavior of the system will be evaluated; this is referred to as test stage. In other words, during the period of time that the behavior of the system is judged to be normal, some events are common, whereas the others are rare. In this method, a buffer consisting of common events is constructed from a stream of data considered to be normal; these are training data. Eventually, during the test stage, if an event does not exist in the buffer, a miss will happen. An anomaly exists in test data provided that the hit rate of the buffer does not reach a predefined threshold. The following subsections include the explanations about the two stages of the proposed method.

4.1.1. The training stage

In this method, due to the fact that the size of the detector window is restricted to the number of n elements, it is endeavored to extract all possible information from data and then to employ them in the test stage. Fig. 3 shows an example of a detector window which has n elements. In this stage, as the stream of training data passes through the detector window, all sequences of size n are scanned. In other words, the stream is broken into contiguous, n-element, overlapping subsequences or n-grams. Duplicate n-grams are removed leaving only the unique ones, and the probability of each unique n-gram is measured. Common sequence is defined to be any sequence of detector window length that occurs in the training data more than a predefined threshold. All common sequences are stored in the buffer for future fast access. This completes the training stage.

4.1.2. The test stage

In this stage, each sequence of size n in the test data is compared element by element to each of n-grams in the buffer as the test data stream passes through the detector window. A *hit* occurs when an unknown n-gram from the test data stream matches one of the stored n-grams in the buffer. Any unknown n-gram that does not match all n-grams of the buffer is termed a *miss*. An anomaly exists in test data provided that the number of consecutive misses exceeds a user-defined threshold.

Fig. 4 illustrates how consecutive misses occur as a sliding window detector is being moved over an anomaly injected into a data stream. In this example, the size of the detector window is 4 ($DW=4$). Suppose that the anomaly type is a foreign sequence of length 6 for which all subsequences of length less than 6 that make up the internal sequences and the boundary sequences are rare (FS RI RB). In this figure, the shaded boxes depict the injected

Fig. 3. Detector window of an anomaly detector.

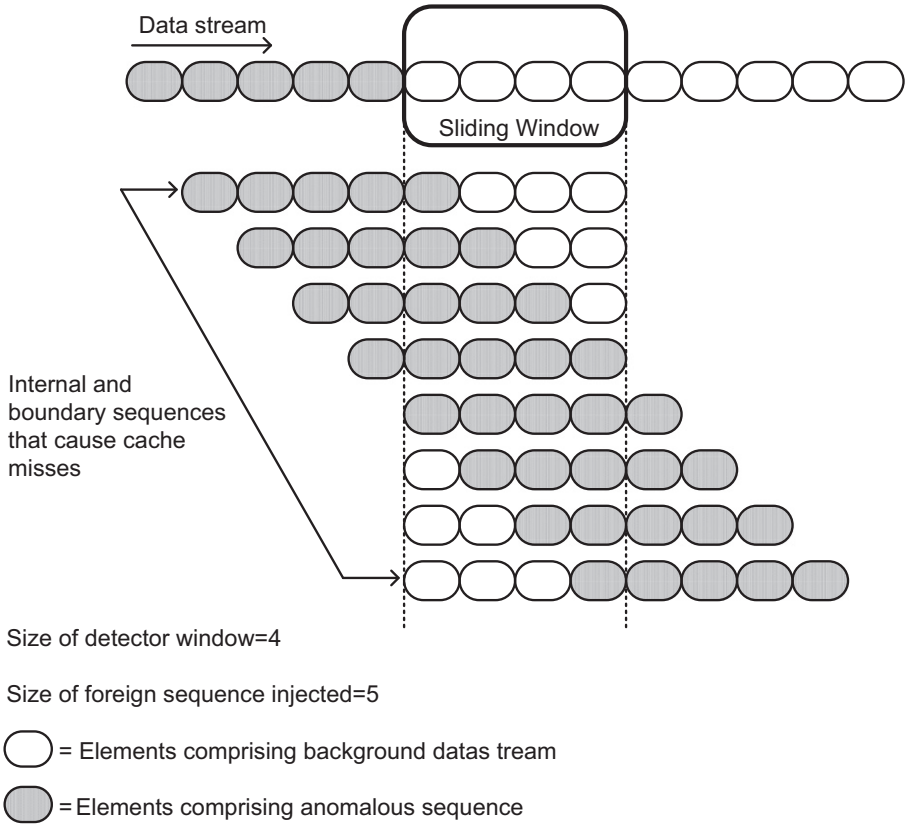

Fig. 4. Foreign sequence anomaly injected into background data.

- Training_stage_algorithm(training data, β, n)

  1) Break training data stream into contiguous, *n*-element, overlapping *n*-grams
  2) Calculate the probability of each unique n-gram
  3) For each unique n-gram do
     2.1) if (probability>β)
          Buffer (n-gram);
     2.2) else
          Discard (n-gram);

- Test_stage_algorithm(test data, buffer, γ, n)
  1) flag=0; counter=0;
  2) While (the end of test data)
     2.1) if (buffer_miss (n-gram) and flag=0)
          counter++; set flag;
        elseif (buffer_miss (n-gram)and flag=1)
          counter++; if (counter > ($\gamma \times n$)) alarm();
        elseif (buffer_hit (n-gram) and flag=1)
          counter--; reset flag;
     2.3) Move *DW* forward.

Fig. 5. Pseudo code of the buffer-based detector.

FS anomaly, which raises an alarm as the number of consecutive misses occurred in the buffer exceeds a user-defined threshold. Nine sequences result from the interaction between the background and the injected anomaly. These nine sequences comprise the rare boundary conditions and rare internal conditions that do not exist in the buffer, the result of which is nine consecutive misses. Therefore, nine misses occur as window slides over the injected anomaly.

The more the number of consecutive misses exceeds a *threshold*, the more anomalous the event will seem. This threshold determines how dissimilar from normal an event can be, while remaining within the bounds of what is considered to be normal behavior. If the threshold of dissimilarity is exceeded, then the observed behavior is judged to be anomalous. Fig. 5 depicts a pseudo code of the buffer-based detector where any sequence of training data stream that occurs more than β percent of time is

considered to be common which is stored in the buffer. Also, γ establishes the minimum distance that encompasses the variation of normalcy. Any unknown event from the test data will be considered anomalous provided that the number of consecutive misses exceeds the *threshold*. Noticeably, the second part of this algorithm represents the test stage of the buffer-based detector. This algorithm detects anomalies based on two variables: *flag* and *counter*, each of which has the value of 0 at the beginning of the algorithm. As the test data is being fed into this detector, any sequence of the detector window length is compared to each of the sequences in the buffer. On one hand, if a miss happens in the buffer, there will be two possibilities at this phase: either the value of *flag* is be equal to zero, which means that at least the previous sequence did not cause a miss in the buffer, so the value of *counter* will be incremented or the variable *flag* contains a non-zero value, which reveals the fact that a number of misses have occurred just before this one in the buffer, hence, the *counter* will be incremented, and its value will be checked against the *threshold*. An anomaly will be detected provided that the incremented value of *counter* exceeds the *threshold*. On the other hand, if the buffer confronts a hit, and the value of *flag* is equal to one, there will be the possibility of being a number of previous misses (flag$=1$) followed by a hit, the result of which is the *counter* being decremented, and the value of *flag* being reset to zero. Then, the algorithm repeats itself till the end of the test data.

4.2. Probabilistic anomaly detector

In this method, using the available information in the sliding window, it is attempted to somewhat evaluate the behavior of the system by means of employing the probability of events. One method to detect the anomaly is through identifying the probability of the subsequent anomalies. Here, sampling of two events with distinct distance is done in order to measure the probability of events in the system; this refers to the training stage. Consequently, during the test stage, the probability of events among a given window size can be measured. An anomaly exists in test data provided that this probability does not reach a predefined threshold. The following subsections include the explanations about the two stages of the proposed method.

4.2.1. The training stage

In this method, the relative distance of symbols to each other is employed to record the behavior of the system in the training stage. As an example, a detector window is shown in Fig. 3, which can see the n-symbol sequence '$d_0\ d_1 \ldots d_{n-1}$' in its purview. It is obvious from this figure that a pair of symbols in this sequence is in the $n-1$ distance, 2 pairs in the $n-2$ distance, 3 pairs in the $n-3$ distance, and therefore $n-1/2$ pairs are orderly in one distance of each other. In this stage, as the stream of training data is passing through the detector window, it is attempted to record the number of pairs of symbols in the sequences of detector window length as well as their distance to each other in order to store their frequency functions. Notably, this information will be stored in the form of a matrix so that by referring to this matrix, test data will be studied.

Considering the matrix constructed during the training stage, the elements of the matrix are $f_i(x, y)$, which stands for the frequency function of the number of a pair of x and y that are in the 'i' distance of each other. Hence, as the stream of training data passes through the detector window, the values $f_i(x,y)$ for each pair of x and y in the alphabet symbols as well as each distance $n-1,\ldots\ 2$ and $i=1$ are measured. To clarify the subject, an example of the constructed matrix obtained from training data is given in Fig. 6. In this figure, the rows of the matrix are all juxtapositions of pairs of symbols from the alphabet containing eight symbols from A to H, and the columns are distance of symbols, which constitutes a pair, varying from 1 to 7 in the detector window of size eight (DW$=8$). The elements of the matrix are $f_i(x, y)$, which stands for the frequency function of the pair of x and y with the distance of 'i'. As an example, f_4(A,B) is the frequency function related to the number of repetitions for the pair A and B that are in the '4' distance of each other. Consequently, the probability of events related to the pair of symbols of x and y that are in the 'i' distance of each other can be measured if the value of each $f_i(x, y)$ is divided by the total number of the frequencies of distance 'i' (Cumulative Distribution Function) in the obtained matrix. This completes the training stage.

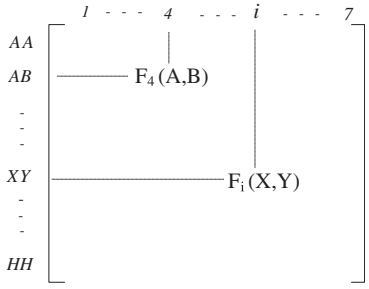

Fig. 6. An example of the constructed matrix obtained from training data.

4.2.2. The test stage

In the test stage, through defining the probability function of the detector window, a value will be obtained based on the relative distance of the symbols in the detector window. The closer this value is to 1, the more normal data will be determined, and the closer this value is to 0, the more anomalous data will be probable. Such function will be defined as there of:

$$p = \prod_{x,y \epsilon \mathrm{DW}} p_i(x,y) \ where\ i = d(x,y) \tag{2}$$

In the function above, DW is a set of symbols in the detector window; besides, n is the number of symbols in the detector window. Furthermore, In the statement $P_i(x,y)$, P is the probability that the pair x and y are in the 'i' distance of each other. This probability is retrieved from the constructed matrix of the training stage. This function employs the product of probabilities to indicate the effect of the probabilities of each value thoroughly and properly. Suppose that *ADEF* is the sequence that a detector window of size 4 can see in its purview. The probability function of the detector window is measured by:

$$P_3(A,F) \times P_2(A,E) \times P_2(D,F) \times P_1(A,D) \times P_1(D,E) \times P_1(E,F) \tag{3}$$

If this sequence is common to the training data, the measured value will be close to "1"; also, this value will be "0" provided that the mentioned sequence is foreign to the training data, and finally the more infrequent this sequence is, the closer this value is to "0". The more this value exceeds a *threshold*, the more anomalous the event will seem. The *threshold* determines how dissimilar from normal an event can be, while remaining within the bounds of what is considered to be normal behavior. If the threshold of dissimilarity is exceeded, then the observed behavior is judged to be anomalous. Fig. 7 depicts a pseudo code of probabilistic detector where γ and β establish the minimum distance that encompasses the variation of normalcy; any event exceeding these thresholds is considered anomalous.

- Training_stage_algorithm(training data, n)

  1) Construct a matrix with *(# of alphabets)²* rows and *(n-1)* columns.
  2) While (the end of training data)
     2.1) for all pairs of symbols in the *DW*that are placed in the *n-1* distance to one distance of each other, record the frequency function in the matrix.
     2.2) Move *DW* forward.
  3) $CDF(i) = \sum_{x,y\ \epsilon\ rows\ of\ matrix} f_i(x,y)\ , i\epsilon[1, n-1]$
  4) Divide each element in column i of the matrix by *CDF (i).*

- Test_stage_algorithm(test data, matrix, β, n)

  1) flag=0; counter=0;
  2) While (the end of test data)
     2.1) $P = \prod_{x,y\epsilon DW} p_i(x,y)\ \ where\ i = d(x,y)$
     2.2) if ($P<\beta$ and flag=0)
          counter++; set flag;
     else if ($P<\beta$ and flag=1)
          counter++;
          if (counter > ($\gamma \times n$)) alarm();
     else if ($P>=\beta$ and flag=1)
          reset counter; reset flag;
     2.3) Move *DW* forward.

Fig. 7. Pseudo code of probabilistic detector.

5. Memory estimation

In this section, an accurate estimation of the number of memory elements required for each detector is done. The required memory was estimated merely for storing the constructed model of normalcy during the training stage. This estimation was calculated when the window sizes of the detectors vary from 2 to 15.

Suppose that $\sum$ is the number of symbols in the alphabet and n is the length of the sliding window. The Markov detector requires memory to store the transition matrix related to the normal behavior of the system. The number of rows of this matrix is equal to the number of states constructed during the training stage. The number of states in the worst-case is equal to $\sum^n$, which are all the possible juxtapositions of $\sum$ symbols in the detector window of size n. Moreover, the number of columns of this matrix is equal to $\sum$ since by shifting the current state one symbol to the left only one new symbol will be added to the right most of the new state. Accordingly, as the detector window slides through training data, only one new symbol will be added to the new state, while the others will remain unchanged. Therefore, the required memory for storing the matrix elements is obtained by:

$$\Sigma^n \times \Sigma = \Sigma^{n+1} \tag{4}$$

The Stide detector requires a database consisting of all unique sequences extracted from the stream of training data. In the worst-case, $\sum^n$ unique juxtapositions of symbols with the size of n are possible, so the required memory is obtained by:

$$\Sigma^n \tag{5}$$

The probabilistic detector requires memory to store the constructed matrix obtained from the training data. The rows of the matrix are all juxtapositions of two symbols of the alphabet, and the columns are distance of symbols varying from 1 to $n-1$. Therefore, the estimated memory is obtained by:

$$\Sigma^2 \times (n-1) \tag{6}$$

The buffer-based detector requires memory to store common sequences scanned from training data stream. The number of common sequences depends on the environmental characteristics, and it is often constant, independent of the length of the sliding window. Let α be the number of common sequences; therefore, the estimated memory is obtained by:

$$\alpha \times n \tag{7}$$

It is interesting to note that the required memory of the Markov and Stide detectors grows exponentially, while the required memory of the buffer-based and probabilistic based detectors grows linearly by the increase in the size of n. Table 1 shows the number of memory elements required for each detector. The numbers are extracted using training data generated in Section 6. It is obvious from this table that the difference between the required memory elements of the proposed detectors and the other detectors becomes significant as n grows from 2 to 15. The required memory of the Markov and Stide detectors approaches the worst-case by the increase in the size of training data. In the case of the buffer-based detector, the obtained result is due to the fact that the number of common sequences is independent of n, and it remains constant as n grows from 2 to 15. In the constructed training data of Section 6, the number of common sequences is 8, independent of the length of the detector window.

6. Constructing the benchmark datasets

This section provides details of the benchmarking process. In general, three kinds of data need to be generated: training data (normal background), testing data (background plus anomalous

Table 1
Number of memory elements required for each detector.

n	Markov [6]		Stide [6]		Probabilistic detector	Buffer-based detector
	# of states	# of memory elements	# of sequences	# of memory elements	# of memory elements	# of memory elements
2	64	491	64	64	64	16
3	491	1042	491	491	128	24
4	1042	2545	1042	1042	192	32
5	2545	4226	2545	2545	256	40
6	4226	6371	4226	4226	320	48
7	6371	8986	6371	6371	384	56
8	8986	12085	8986	8986	448	64
9	12085	15675	12085	12085	512	72
10	15675	19749	15675	15675	576	80
11	19749	24310	19749	19749	640	88
12	24310	29385	24310	24310	704	96
13	29385	34955	29385	29385	768	104
14	34955	40993	34955	34955	832	112
15	40993	327944	40993	40993	896	120

events), and the anomalies themselves. In benchmarking parlance, the training and testing data constitute the anomaly detector workload.

6.1. Generating the training and test data

The training data serve as the "normal" data into which anomalous events are injected. The requirements for the training data are that a large portion of the data should be comprised of common sequences, that they contain a small proportion of rare sequences, and that there has to be a relatively high predictability from one symbol to another.

The alphabet has eight symbols: *A, B, C, D, E, F, G,* and *H*. Since increasing the alphabet size would not change the outcome, maintaining a relatively small alphabet size facilitates a more manageable experiment, yet permits direct study of detector response. The training data were generated from an eight by eight state transition matrix with the probability of 0.9672 in one cell of each row, and 0.004686 in every other cell, resulting in a sequence of conditional entropy 0.1 [25]. One million data elements were generated so that there would be a sufficient variety of rare sequences for the test data. 98% of the training data consisted of the repetitions of the sequence *ABCDEFGH*, seeding the data set with common sequences. This is the data set applied to train the detector used in this study, i.e. to establish a model of normalcy against which unknown data can be compared.

Test data, containing injected anomalies, are used to determine how well the detector can capture anomalous events and correctly reject events that are not anomalous. The test data consist of two components: a background, into which anomalies are injected, and the anomalies themselves. Each is generated separately, after which the anomalies are injected into the background under strict experimental control. The background is the same as the training data generated in the previous stage.

6.2. Constructing the anomalous injections

Once the background data are available, anomalous events must be injected into them to finalize the test data. The selected anomaly type is a foreign sequence of length AS for which all subsequences of length less than AS that make up the internal sequences and the boundary sequences are rare. Rare is defined to be any sequence of detector window length that occurs in the training data less than 1% of the time.

Once that anomaly type is determined, e.g. FS RI RB, as described in section 2, the next step is to inject a foreign sequence composed of rare sequences into the test data. A catalog of rare *n*-grams is obtained from the training data. Rare *n*-grams are drawn from the catalog and composed to form a foreign sequence of the appropriate size. Sufficient numbers of rare *n*-grams are added to the constructed anomaly to form the rare boundary conditions as the anomaly passes through the detector window. For example, the sequences of size three *BAF*, *AFH, FHE, HEC, ECC,* and *CCF,* each of which occurred less than 1% of the time in the training data; consequently, these are rare sequences. Combining these six sequences (*BAFHECCF*) produces one anomaly of size four (*FHEC*) whose internal sequences and boundary conditions are made up of rare sequences of size three. This anomaly was injected into the background data.

Eight injection sizes and fourteen detector window sizes have been tested. The procedure outlined above for creating the anomalous events and for injecting them, is repeated for each combination of injection size and window size, resulting in 112 total data sets.

7. Experimental results

This section describes the experiments performed to evaluate the detection coverage of each method. Furthermore, the area and delay of each detector is measured using a synthesis tool.

7.1. Detection coverage

Each of the four detectors, Markov, Stide, buffer-based, and probabilistic-based detector was implemented by C++ program and provided with the same set of training data. From the training data, the detectors learned their models of normal behavior. Then, each detector was tested using each of the 112 test data sets described in section 6. The size of the detector window was varied from 2 to 15, and the size of the injected event was varied from 2 to 9.

Detection and blind regions for each method are depicted in Figs. 8–11. These decision maps illustrate the detection capability of each method with respect to an injected foreign sequence composed of rare sequences. The *x*-axis of each map marks the increasing size of the foreign sequence injected into the test data;

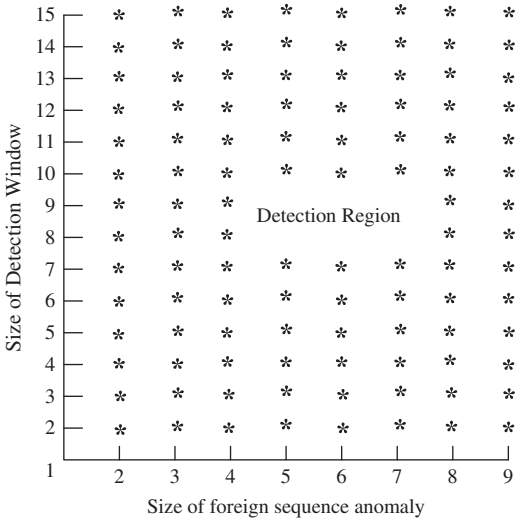

Fig. 8. Detection coverage of Markov detector [6].

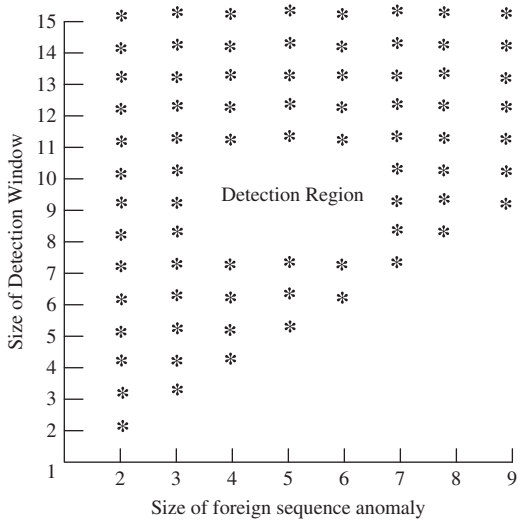

Fig. 9. Detection coverage of Stide detector [6].

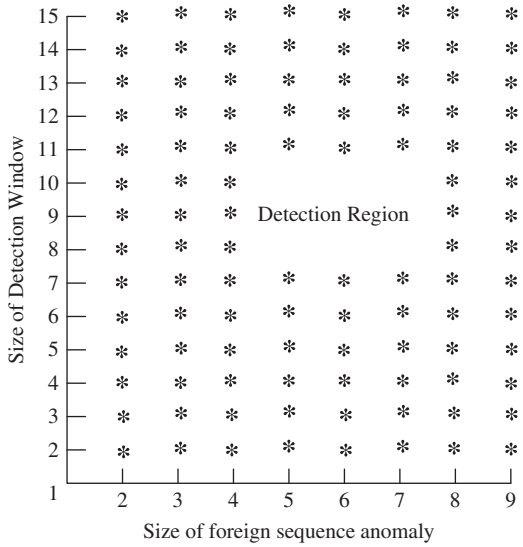

Fig. 10. Detection coverage of buffer-based detector.

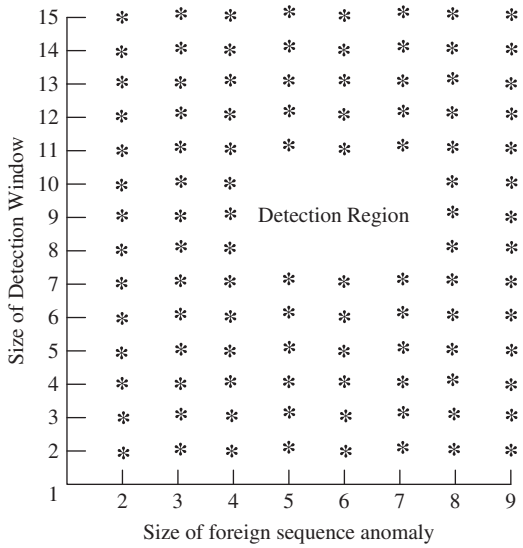

Fig. 11. Detection coverage of Probabilistic detector.

the *y*-axis marks the size of the detector window required to detect a foreign sequence of a specified size. Each star mark indicates successful detection of the foreign sequence anomaly whose size is indicated on the *x*-axis through using a detector window whose size is marked on the *y*-axis. The areas that are bereft of stars indicate detection blindness, which means that the detector was unable to detect the injected foreign sequence whose corresponding size is marked on the *x*-axis.

The results show that although all detectors use the concept of a sliding window, their different similarity metrics significantly impact their detection capabilities. In the case of Stide, even though there is a foreign sequence present in the data stream, it is visible only if the size of the detector-window is at least as large as the foreign sequence- a requirement that the Markov detector and the proposed detectors do not have. These figures show that the proposed detectors, and the Markov detector have 100% detection capability, yet Stide detector has only 75% detection coverage; moreover, the following subsection shows that the area overhead for the proposed detectors is significantly less than the Markov and Stide detector.

7.2. Hardware implementation

Each of the four detectors, Markov, Stide, buffer-based, and probabilistic was implemented by C++ program, and then using VHDL code generator, VHDL code of each detector was produced. The performance of each detector was examined using Modelsim simulation tool to make sure that they all perform properly. Eventually, Leonardo Spectrum synthesis tool was used in order to transform the hardware description of each detector to the corresponding logic/electronic description. Due to the large capacity of the anomaly detectors, implementation of these systems on small FPGA boards was impossible; therefore, ASIC technology was used for synthesis of these detectors. In order to evaluate the area and delay of each anomaly detector, a number of VHDL files produced by code generator were fed to the synthesis tool, when the window sizes of the detectors vary from 3 to 6.

Table 2 shows a comparison between the Area and delay of each detector when the window sizes of the detectors vary from 3 to 6. As it can be seen in this table, the area of Markov and Stide detectors grows, respectively, 739% and 910%, by the increase in the size of detector window, while the area of buffer-based and probabilistic detectors grows 51% and 360%, respectively. Moreover, the delay of the probabilistic method is quite competitive with Stide and Markov detectors for all window sizes. It is important to note that the greater value of delay for the buffer-based detector originates from the nature of buffer in this method. The results show that the efficiency of Markov and Stide detectors in comparison with buffer-based and probabilistic detectors decreases enormously with the growth in the size of detector window.

7.3. Efficiency

Table 3 illuminates a comparison between different methods presented in this paper in terms of their performance when the size of the detector window varies from 3 to 6. The overall performance of each method is composed of three components that are detection coverage, area overhead, and delay overhead; therefore, the performance will be defined as there of:

$$Performance = \alpha \times Detection\ coverage + \frac{\beta}{Delay\ overhead} + \frac{\gamma}{Area\ overhead}. \quad (8)$$

Table 2
A comparison between area and delay of the anomaly detectors.

Anomaly detector	DW=3		DW=4		DW=5		DW=6	
	Delay (ns)	Area (μm^2)	Delay (ns)	Area (μm^2)	Delay (ns)	Area (μm^2)	Delay (ns)	Area (μm^2)
Markov [6]	2.6	316	2.6	434	2.6	974	2.6	2652
Stide [6]	2.3	30	3	64	2.5	186	2.9	303
Buffer-based	4.2	80	4.2	91	5	100	4.2	121
Probabilistic	2.5	73	3.4	107	0.7	164	1.9	336

Table 3
Performance of different anomaly detectors.

Anomaly detector	DW=3	DW=4	DW=5	DW=6
Markov [6]	**46.25**	**46.23**	**46.18**	**46.16**
Stide [6]	**40.35**	**36.38**	**38.26**	**36.35**
Buffer-based	**41.69**	**41.64**	**40.33**	**41.54**
Probabilistic	**47.12**	**43.44**	**81.15**	**50.97**

Table 4
Training and Anomalous data sets (DW=5,3 $\leq AS \leq$ 9).

	AS < DW		*AS*=DW		*AS* > DW		Total
	#	(%)	#	(%)	#	(%)	
Training data sets	**0**	**0**	**0**	**0**	**0**	**0**	**2000**
Anomalous data sets	**700**	**35**	**600**	**30**	**700**	**35**	**2000**

The Detection coverage of each detector has been directly extracted from Figs. 8–11, while the delay and the area overheads have been retrieved from Table 2 of the previous subsection. In the function above, α, β, and γ are user-defined constants, each of which varies from "0" to "1", while the summation of α, β, and γ is always equal to "1". Note worthily, these parameters determine the relative importance of the performance components in multifarious environments. For instance, the greater value chosen for γ verifies the tremendous importance of area overhead among all performance components in that specific environment. Generally speaking, the value "1/3" has been chosen for all of these constants in order that these components play the equal role in our experiments. As it can be seen in Table 3 in which the overall performance of each method for four window sizes from 3 to 6 is presented, the probabilistic method with the *Detector Window* of 5 has shown the best performance among all methods. Since α, β, and γ are situation-specific and depends on environmental characteristics, different values for these parameters can be found more appropriate in various environments.

7.4. Effects of parameters on detection coverage

As it is mentioned in previous sections, there are three different types of anomalies including foreign symbols, foreign sequences and rare sequences. In the case of foreign symbol anomalies, it is obvious that all the traditional and the proposed methods can detect them since the methods construct their models of normal behavior based on the training data alphabets; therefore, detecting a foreign symbol that generates foreign n-grams in the anomalous data set is straightforward. Another type of anomaly which is called rare sequence is very similar to foreign sequence anomalies in the way which they can be detected since for generating a foreign sequence anomaly several rare sequences should be concatenated so that the whole concatenated sequence would be foreign. Therefore, any detector that is able to detect a foreign sequence anomaly which is composed of several rare sequences can definitely detect rare sequence anomalies, which are also comprised of several rare sequences.

Consequently, foreign sequences anomalies are the most important type of anomalies in order to test different methods of anomaly detection. Hence, about 2000 training data sets with the size of 5000 symbols as well as 2000 anomalous data sets whose types are foreign-sequence with the size of 5000 symbols were generated; also, the alphabet is comprised of 8 symbols. In order to generalize the experiments, a detector window with the length 5 is selected, and the length of foreign sequence anomaly varies between 3 and 9. About respectively 35%, 30%, and 35% of the anomalous data sets are generated in such a way that the size of the anomaly (AS) is less than, equal to, and more than the size of detector windows. Table 4 illustrates the parameters pertaining to our experiments.

All four methods were evaluated using the generated data sets. Table 5 shows the results of each method for different types of data sets. Noticeably, the parameters used for each method are shown in the second column. The results of each experiment are specified in terms of *False Positive, False Negative, True Positive, and True Negative.*

It is worth mentioning that regarding Stide and Markov detectors, the constant rate of false alarms (*False Positive)* which occurs at the starting point of data sets is due to the system warm up. It should also be noted that the main objective for all anomaly detection methods is to determine whether the test data is anomalous or not independent of where and what type of anomaly it contains, so false alarms would be taken into account only in non-anomalous data sets or before detecting an injected anomaly. Accordingly, these false alarms would be generated again when anomalous data sets are fed into Stide and Markov detectors. However, for getting a better observation, the warm-up alarms have been excluded from anomalous data sets, and the coverage after warm-up is shown. As it can be seen, Markov method can detect 100% of anomalies while Stide detector is only able to detect anomalous data sets whose anomaly sizes are less than the length of the detector window.

Moreover, regarding Buffer-based and Probabilistic methods, the rates of false positives for training data sets are less than 5% which is very low in comparison with the other detectors. Note worthily, these rates are highly dependent to the parameters chosen for these methods. A large proportion of training data sets were fed into the detectors; also, the parameters were selected in such a way that the rate of false positives will become low. However, the detection coverage of these methods for anomalous data sets is more than 90% which is noticeably high. This can be increased by selecting the parameters under more strict conditions (see Buffer method), yet the rate of false positives would also increase.

Table 5
Results of anomaly detection of different mechanism with some parameters.

Method	Method parameter	Training data sets (Total of 2000 samples)				Anomalous data sets (Total of 2000 samples)			
		False Positive		True Negative		True Positive		False Negative	
		#	(%)	#	(%)	#	(%)	#	(%)
Markov	**Surprise factor=0.01**	**2000**[a]	**100**	**0**	**0**	**2000**	**100**	**0**	**0**
Stide	**–**	**1910**[a]	**95.5**	**90**	**4.5**	**1300**[b]	**65**	**700**[c]	**35**
Buffer-based	**β=0.01 γ=20**	**100**	**5**	**1900**	**95**	**1984**	**99.2**	**16**	**0.8**
Probabilistic	**β=1E-10 γ=20**	**15**	**0.75**	**1885**	**94.25**	**1800**	**90**	**200**	**10**

[a] These cases are due to system warm up.
[b] In this case, the anomalous data sets contain anomaly size *AS*, $3 \leq AS \leq 5$.
[c] In this case, the anomalous data sets contain anomaly size *AS*, $6 \leq AS \leq 9$.

8. Conclusions

This paper presented two detection methods namely buffer-based detector and probabilistic detector. The former constructs a cache consisting of events from a stream of data considered to be normal. Consequently, during test stage, if an event does not exist in the cache, a miss will happen. An anomaly exists in test data provided that the hit rate of the cache does not reach a predefined threshold. The latter employs the probability of events to evaluate the behavior of system. Sampling of two events with distinct distance is done in order to measure the probability of events in the system; this refers to the training stage. Eventually, during test stage, the probability of events among a given window size can be measured. An anomaly exists in test data provided that this probability does not reach a predefined threshold. The experiments on 112 standard benchmarks show that the proposed methods can detect 100% of anomalies. Also, the area overhead of the proposed detectors grows linearly, while the area overhead of other typical detectors grows exponentially by the increase in the size of the detector window. This point is very essential for embedded systems which have restrictions on required hardware.

References

[1] M. Short, M.J. Pont, Assessment of high integrity embedded automotive control systems using hardware in the loop simulation, J. Syst. Softw. 81 (7) (2008) 1163–1183.
[2] A.V. Lovato, E. Araujo, and J.D.S. da Silva, Fuzzy decision in airplane speed control In: Proceedings of the IEEE International Conference on Fuzzy Systems, pp. 1578–1583, 2006.
[3] R. Bastide, D. Navarre, P. Palanque, A. Schyn, and P. Dragicevic, A model-based approach for real-time embedded multimodal systems in military aircrafts, In: Proceedings of the ACM international conference on Multimodal Interfaces, pp. 243–250, 2004.
[4] H. Al Nahas and J.S. Deogun, Radio frequency identification applications in smart hospitals, In: Proceedings of the IEEE International Symposium on Computer-Based Medical Systems, pp. 337–342, 2007.
[5] A. Scholz, S. Sommer, C. Buckl, G. Kainz, A. Kemper, A. Knoll, J. Heuer, and A. Schmitt, Towards and adaptive execution of applications in heterogeneous embedded networks, In: Proceedings of the ACM/IEEE International Workshop on Software Engineering for Sensor Network Applications, 2010.
[6] R.A. Maxion, K.M.C. Tan, Anomaly detection in embedded systems, J. IEEE Trans. Comput. 51 (2) (2002) 108–120.
[7] V. Chandola, A. Banerjee, V. Kumar, Anomaly detection: a survey, ACM Comput. Surv. 41 (3) Article 15.
[8] V. Kumar, Parallel and distributed computing for cyber security, J. Distrib. Syst. Online 6 (2005).
[9] C. Spence, L. Parra, and P. Sajda, Detection, synthesis and compression in mammographic image analysis with a hierarchical image probability model In: Proceedings of the IEEE International Workshop on Mathematical Methods in Biomedical Image Analysis, pp. 3–10, 2001.
[10] E. Aleskerov, B. Freisleben, and B. Rao, Cardwatch: A neural network based database mining system for credit card fraud detection, In: Proceedings of the IEEE International Conference on Computational Intelligence for Financial Engineering, pp. 220–226, 1997.
[11] R. Fujimaki, T. Yairi, and K. Machida, An approach to spacecraft anomaly detection problem using kernel feature space, In: Proceedings of the ACM SIGKDD International Conference on Knowledge Discovery in Data Mining, pp. 401–410, 2005.
[12] M. Bicego, V. Murino, M. Pelillo, A. Torsello, Similarity-based pattern recognition, J. Pattern Recognit 39 (10) (2006) 1813–1824.
[13] R.A. Maxion, F.E. Feather, A case study of Ethernet anomalies in a distributed computing environment, IEEE Trans. Reliab. 39 (04) (1990) 433–443.
[14] S. Boriah, V. Chandola, and V. Kumar, Similarity measures for categorical data: a comparative evaluation, In: Proceedings of the SIAM International Conference on Data Mining, pp. 243–254, 2008.
[15] S. Budalakoti, A. Srivastava, and M. Otey, Anomaly detection and diagnosis algorithms for discrete symbol sequences with applications to airline safety, In: Proceedings of the IEEE International Conference on Systems, Man, and Cybernetics, pp. 101–113, 2007.
[16] D. Janakiram, V. Reddy, and A. Kumar, Outlier detection in wireless sensor networks using bayesian belief networks, In: Proceedings of the IEEE International Conference on Communication System, Software and Middleware, pp. 1–6, 2006.
[17] J. Branch, B. Szymanski, C. Giannella, R. Wolff, and H. Kargupta, In-network outlier detection in wireless sensor networks, In: Proceedings of the IEEE International Conference on Distributed Computing Systems, pp. 51–58, 2006.
[18] T.V. Phuong, L.X. Hung, S.J. Cho, Y. Lee, S. Lee, An anomaly detection algorithm for detecting attacks in wireless sensor networks, J. Intell. Secur. Inf. 3975 (2006) 735–736.
[19] W. Du, L. Fang, and N. Peng, Lad: localization anomaly detection for wireless sensor Networks, In: Proceedings of the IEEE international symposium on parallel and distributed processing, pp. 41a, 2006.
[20] S. Subramaniam, T. Palpanas, D. Papadopoulos, V. Kalogeraki, and D. Gunopulos, Online outlier detection in sensor data using non-parametric models, In: Proceedings of the ACM international conference on Very large data bases, pp.187–198, 2006.
[21] K. Zhang, S. Shi, J. Li, Unsupervised outlier detection in sensor networks using aggregation tree, J. Adv. Data Min. App. 4632 (2007) 158–169.
[22] T. Ide, S. Papadimitriou, and M. Vlachos, Computing correlation anomaly scores using stochastic nearest neighbors In: Proceedings of the IEEE International Conference on Data Mining, pp. 523–528, 2007.
[23] V. Chatzigiannakis, S. Papavassiliou, M. Grammatikou, and B. Maglaris, Hierarchical anomaly detection in distributed large-scale sensor networks In: Proceedings of the IEEE international Symposium on Computers and Communications, pp.761–767, 2006.
[24] S. Jha, M.C. Tan, and R.A. Maxion, Markov chains, classifiers, and intrusion detection, In: Proceedings of the IEEE International Workshop on Computer Security Foundations, pp. 206–219, 2001.
[25] R.A. Maxion and K.M. Tan, Benchmarking anomaly-based detection systems, In: Proceedings of the IEEE International Conference on Dependable Systems and Networks, pp. 623–630, 2001.
[26] M.C. Tan and R.A. Maxion, Why 6? Defining the operational limits of stide, an anomaly-based intrusion detector, In: Proceedings of the IEEE Symposium on Security and Privacy, pp. 188–201, 2002.
[27] D. Mutz, F. Valeur, C. Kruegel, G. Vigna, Anomalous system call detection, ACM Trans. Inf. Syst. Security 9 (1) (2006).
[28] J.M. Agosta, C. Diuk-Wasser, J. Chanderashekar, C. Livadas, An adaptive anomaly detector for worm detection, In: Proceedings of the USENIX workshop on Tackling computer systems problems with machine learning techniques, pp. 1–6, 2007.
[29] M.Q. Ali, H. Khan, A. Sajjad, S.A. Khayam, On Achieving Good Operating Points on an ROC Plane using Stochastic Anomaly Score Prediction, ACM Conference on Computer and Communications Security (CCS), pp. 314–323, November 2009.