Compact Fault Dictionaries for Efficient Sensor Fault Diagnosis in IoT-enabled CPSs

Stavros A. Viktoros, Maria K. Michael, Marios M. Polycarpou
Dept. of Electrical and Computer Engineering, KIOS Research and Innovation Centre of Excellence, University of Cyprus
Email: svikto01@ucy.ac.cy, mmichael@ucy.ac.cy, mpolycar@ucy.ac.cy

Abstract—The recent advances in the area of Internet-of-Things (IoT) have allowed for the implementation of complex large-scale Cyber-Physical Systems (CPSs). This phenomenon calls for efficient and scalable solutions for the new challenges being introduced. Sensor fault diagnosis has emerged as a priority in various IoT-enabled CPSs, especially for critical infrastructure applications where multiple IoT devices might be in use. In this work, we examine the problem of building a compact fault dictionary which allows for efficient real-time model-based multiple sensor fault detection and isolation. The problem under consideration is formulated as a combinatorial set problem and then efficiently encoded using Zero-suppressed binary Decision Diagrams (ZDDs), which are specialized data structures based on Boolean theory. The proposed approach is highly scalable with respect to the total number of sensor fault scenarios considered. Using the respective ZDD as a fault dictionary reduces the memory requirements by several orders of magnitude when compared to the conventional approach. This is achieved while allowing the fault isolation process to occur in linear time to the size of the dictionary. Our experimental results show that it takes between 0.002s to 0.012s for performing the fault isolation process in the range of tested systems.

Index Terms—Sensor fault Detection and Isolation, Fault Dictionaries, Internet-of-Things, Cyber-Physical Systems, Decision Diagrams

I. INTRODUCTION

In the era where the Internet-of-Things (IoT) is being realized and Machine-to-Machine communication is becoming a standard practice, new problems and challenges arise. Leveraging the IoT advances to enhance automation, engineers continue to design and build Cyber-Physical Systems (CPSs) in an increasing scale. A CPS consists of two main layers; the physical layer which can host physical, biological or engineered systems with IoT devices, and the cyber layer which in addition to the communication network it also provides the computational power to monitor and control physical layer components [1]. CPSs will continue to grow in scale and complexity as the advances in communication technologies and the Internet-of-Things move forward.

In the case of CPSs, process automation becomes feasible after the collection of enough data through multiple sensors. Thus, one of the most important challenges introduced is maintaining the integrity of the vast amount of data produced from sensor measurements. Faulty behavior of a key element such as the sensors [2], caused by either a transient/permanent fault or a malicious act, can have a significant impact in an unsupervised environment. That is why monitoring and control is a critical part of a CPS, especially when data is gathered from IoT devises. Fault detection refers to the process of determining the presence of faults in a system while fault isolation deals with locating the faults. The Fault Detection and Isolation (FDI) problem, may also be referred to as Fault Diagnosis.

While various general fault diagnosis methodologies exist [3], detecting and especially isolating multiple sensor faults emerges as a significantly challenging problem in large-scale CPSs [17]. This is due to the large number of sensors and sensor networks utilized to provide data for coordinating the interactions between the two layers. This problem is exacerbated by the possibility of multiple sensor faults, leading to a huge number of fault scenario combinations.

To enable multiple sensor fault diagnosis in CPSs, works [4], [5] follow an approach which utilizes observers for detecting the faults. In the same direction, and to address scalability, [6] groups sensors into local sensor sets. This allows for a single observer to be used for a group of sensors. In this case, the isolation process can be handle by an aggregation module which process the data from all observers. Fault dictionaries can be an efficient solution for processing this data and realizing multiple sensor fault isolation [18]. A fault dictionary provides the system with the knowledge of all possible sensor fault signatures and, thus, enables the fault isolation between multiple sensor faults. However, due to the large number of possible fault scenario combinations, the size of this dictionary can grow exponentially to the size of the underlying fault model and, therefore, render the entire process impracticable.

This work focuses on the formulation and efficient generation of a fault dictionary to enable multiple sensor fault isolation in large-scale CPSs. The main contribution of this work is the generation of a highly compact fault dictionary, which allows for real-time fault isolation as the time required by the isolation process grows linearly to the number of sensors in the system. We demonstrate the impact of the proposed approach when integrated in a centralized architecture, as described in the next section. Hence, we are able to show in a quantitative manner the advantages of the proposed method when compared to conventional methods for fault dictionary based FDI.

For the generation of the fault dictionary, we propose using Zero-suppressed binary Decision Diagrams (ZDDs) [7], which are variants of the widely known Binary Decision Diagrams (BDDs) [8]. While BDDs are mainly used for problems formulated based on Boolean Algebra, ZDDs are mostly beneficial...
for binary logic based combinatorial set representation and manipulation. ZDDs are versatile in nature and have been adapted in a plethora of automation applications including electronic and test automation [10], [11], pattern recognition for DNA sequences [13], speech recognition [14], etc. In this work, we first formulate the considered fault diagnosis problem into a combinatorial set problem and subsequently we show how to efficiently build and manipulate the corresponding ZDD to perform fault isolation.

The resulting ZDD is able to compactly represent the conventional fault dictionary, thus significantly reducing the memory requirements. This allows saving the entire fault dictionary in the memory of the monitoring and/or aggregator agents in an IoT environment. Another major benefit of using the ZDD data structure, is the linear to its size access time. This is proven from our experiments where a number of systems were tested. The results show that a ZDD-based fault dictionary can reduce the memory footprint by several orders of magnitude while allowing for real-time fault isolation. The proposed approach is applicable to FDI approaches utilizing fault dictionaries for various IoT devices, as long as the measurements from the devices generate information which can be encoded by a binary function.

The rest of the paper is organized as follows. Section II gives the necessary preliminary information regarding our work. Section III describes the proposed formulation and generation of the fault dictionary using ZDD and Section IV demonstrates how the ZDD-based fault dictionary can be utilized for fault isolation. Section V presents the experimental results and Section VI concludes the paper.

II. PRELIMINARIES

A. Underlying Fault Detection and Isolation Scheme

Consider a typical Cyber-Physical System Architecture as shown in Figure 1, where sensors are used to provide information/measurements from physical components of the system and the cyber layer provides monitoring and control of the system. In such systems, sensor measurements are of critical importance. While detecting a single sensor fault has a difficulty of its own, isolating faulty sensors under the presence of multiple faults is even more challenging. To enhance multiple sensor fault detection and isolability, sensors are grouped into small local sensor sets that can be monitored through dedicated monitoring agents.

Sensor grouping helps in addressing multiple issues that can appear in large-scale CPSs. The first issue concerns the scalability. Even though it could be possible to have a monitoring module per sensor, this comes with additional overhead on overall hardware cost. Secondly, a very important benefit is maintaining system observability. Fault diagnosis approaches which utilize observers, usually detect faults based on residual estimations. This means that an estimation for the correct state of the system is generated and compared to the current state. There are cases where a single sensor measurement might not be able to accurately describe the state of the system. Thus, by grouping a number of sensors together we can reduce the possibility of encountering an unobservable state. Lastly, it can help in improving the detection algorithm used. This is achieved through sharing sensor measurements across the different sensor sets, which allows for an improved detection logic. All of the above are discussed in [6]. For the purposes of this work, the grouping procedure is done in a random manner to provide us with the flexibility to create multiple configurations.

Figure 2a shows communication interconnections required for performing sensor fault detection and isolation. Let us assume a system with a global sensor set $S$ comprised of $n$ sensors. $S$ is decomposed into $q$ local sensor sets, where each set $S_i$ can contain one or more sensors. A local set may be disjoint, which means all of its sensors are unique to it, or overlapping with other sets which means one or more sensors are common between one or more sets. A monitoring agent $M$ is then deployed which includes $q$ monitoring modules and the Aggregation module $A$. Each monitoring module $M_i$, is assigned to one local sensor set and is responsible for detecting faults within that set. A decision $d_i$, is generated from each of these modules to express the presence or absence of a fault. If a fault is observed in a local sensor set then the respective decision value is 1, otherwise it is 0. The decisions are then combined into a string to form the observed fault pattern $D$. Lastly, the aggregation module $A$ will receive the observed fault pattern and execute the fault isolation process. In our case, this is done with the aid of a Fault Dictionary which includes all possible sensor fault signatures, deriving by the underlying fault model. By the end of the fault isolation process, the aggregation module is responsible for delivering the isolation data $I$.

Figure 2b shows an example where $S = \{s_1, s_2, s_3, s_4\}$. Assume that after the decomposition process, four local sensor sets are created. $S_1 = \{s_1\}, S_2 = \{s_2, s_3\}, S_3 = \{s_3, s_4\}$ and $S_4 = \{s_4\}$. Consequently, $M$ will have four monitoring modules whose decisions will be processed by $A$. This example will be referenced multiple times throughout the paper.

B. Sensor Fault Signatures

Depending on the targeted fault model, a set of sensor fault scenarios has to be considered in order to formulate the sensor
fault signatures. The fault scenarios might include single or multiple sensor faults. Each of these faults will trigger a different response from the monitoring modules. A monitoring module will set its decision value to 1 only if it detects a fault. A fault pattern is then formed with all the decision values and associated with the respective fault scenario. We refer to this as the Sensor Fault Signature.

Sensor fault signatures, and subsequently the fault dictionary, can be formulated once the local sensor sets are formed. The total number of fault scenarios \( f_{\text{tot}} \), can be calculated with respect to the cardinality of multiple sensor faults considered by the fault model, as shown below:

\[
f_{\text{tot}} = \frac{f_m}{r! (n - r)!}
\]

where \( f_m \) is the multiple sensor faults cardinality and \( n \) is the number of sensors.

In approaches such as [6] [18], fault signatures are stored in a conventional 2D-array. Table I shows an example of this approach with the fault dictionary for the reference system of Figure 2b when all possible sensor fault scenarios are considered. Based on equation 1, we will have a total of 15 fault scenarios \( F_{S_{\text{tot}}} \), where each scenario corresponds to a different multiple sensor fault (including single faults). For example, \( F_{S_1} \) is the scenario with single sensor fault in \( s_1 \), \( F_{S_2} \) is the scenario with multiple sensor fault scenario in \( s_1, s_2 \), and so on. Each column in the array is a different fault scenario and each row represents the expected decision values \( D_q \) of the corresponding monitoring module for the different fault scenarios.

C. Zero-suppressed binary Decision Diagrams

In this work, we use Zero-suppressed binary Decision Diagrams (ZDDs) to build the Fault Dictionary in order to avoid the exponential growth of the typical approach as previously described. ZDDs are variants of the well known Binary Decision Diagrams (BDDs) proposed in the seminal works of [8], [9], a specialized data structure based on Boolean Theory which can be seen as graphical representations of Boolean functions. Each node corresponds to a boolean variable and has two types of possible outgoing edges. The "if" edge which indicates that the variable value is 1 and the "else" edge for variable value 0.

See for example the BDD in Figure 3a which corresponds to the function \( f = ((a \times b) + (a \times c) + (b \times c)) \), with solid lines representing the "if" and dashed lines the "else" edges. Every BDD has two constant value terminal nodes (0 and 1), for the two possible values of a boolean function. They follow a variable order forced by the designer and they are levelized which means that each level may include nodes of only one variable based on that order. This is a key attribute of BDDs since it makes them unique representations for a given boolean function for a specific variable order. The size of a BDD is depended on the number of variables and variable ordering.

\[
\text{ZDDs} \text{ [7] have been shown to be particularly efficient and able to compactly represent combinatorial sets. Both BDDs and ZDDs use reduction rules in order to suppress unnecessary nodes and reduce the size of the diagram. In BDDs a node suppression implies that the node takes the don’t care value in the combination. In ZDDs, node suppression happens if a node with the value of 1 leads to the terminal node 0. For example, see Figure 3b which corresponds to the combinatorial set } s = \{ab=c, a=bc, abc\}. \]

Fig. 3. (a) BDD representation of the function \( f = ((a \times b) + (a \times c) + (b \times c)) \) (b) ZDD representation of the equivalent combinatorial set \( s = \{ab=c, a=bc, abc\} \).
especially efficient in representing sparse combinatorial sets.

TABLE II
STANDARD ZDD OPERATIONS USED FOR GENERATING AND MANIPULATING THE ZDD BASED FAULT DICTIONARY.

| Function         | Symbol | Description                                |
|------------------|--------|--------------------------------------------|
| Union(T,F)       | $F \cup T$ | Calculates the union of the two sets |
| Intersect(T,F)   | $F \cap T$ | Calculate the intersection of the two sets |
| Change(F,i)      | $\chi_i(F)$ | Invert $i$-th variable in $F$ |
| SubSet0(F,i)     | $C^0_i(F, i)$ | Cofactor of $F$ when $i = 0$ |
| SubSet1(F,i)     | $C^1_i(F, i)$ | Cofactor of $F$ when $i = 1$ |
| Product(T,F)     | $F \times T$ | Computes the product of two ZDDs |
| Essential(F)     | $\xi(F)$ | Cube of the essential variables in $F$ |
| DagSize(F)       | $|F|$ | # ZDD Nodes in $F$ |
| Empty()          | 0      | Empty Set |

For generating and manipulating the ZDD-based fault dictionary, we built our framework using the CUDD package [15], an academic library, which implements a plethora of standard operations used in Boolean and Set Theory that are necessary for building the ZDD. A list of all the functions used in our framework is shown in Table II.

III. ZDD-BASED FAULT DICTIONARY

A. Problem Formulation

For the fault dictionary to be represented using a ZDD, the considered FDI problem is formulated as a combinatorial set problem. Sensor Fault signatures, as described in Section II-B, are used to formulate the combinatorial sets. For better understanding how the fault signatures can be adapted, the first 3 columns of Table I are shown in the form of a truth table in Table III, where each $d_i$ variable corresponds to a monitoring module decision, each $s_i$ variable to a sensor and $F$ is the function that models the system state, faulty or fault-free. These are also the variables that will be used to generate the corresponding ZDD.

It is important to note that the truth table is not necessary for generating the ZDD but it is rather used only for visualizing the problem formulation. Instead, the minterms are generated dynamically, based on two different approaches, which will be discussed in Section III-B.

The rationale behind this adaptation is that each minterm corresponds to one fault signature or else, a sensor fault scenario along with the fault pattern that will generate. For the example shown in Table III the 4 Most Significant Bits (MSBs) represent the fault pattern, while the 4 Least Significant Bits (LSBs) form the sensor fault scenario. That means, the first line models the fault pattern $D(t) = [0, 0, 0, 1]$, which corresponds to the first column of the fault dictionary as shown in Table I, and indicates that only sensor $s_1$ is faulty, which will result to $I(t) = \{FS_1\}$.

B. Fault Dictionary Generation

The ZDD-based fault dictionary generation process is implemented with two different approaches. The Enumerative approach as shown in Algorithm 1 and the Non-Enumerative approaches as shown in Algorithm 2. Both algorithms require three inputs, the number of sensors, local set information and the fault model or else the multiple sensor fault cardinality desired. The first is used to initialize the global sensor set, while the second is used for calculating all $D_i$ variables. The fault model is used to set the multiple sensor fault cardinality. That essentially means how many sensors might be faulty at the same time. This functionality is added in order to be able to generate fault dictionaries for fault models that do not consider all the possible faults.

1) Enumerative Approach: Consider $v$ as the variable set such that $v = \{S \cup D\}$, where $S = \{s_1, s_2, ..., s_n\}$ is the sensors variable set while $D = \{d_1, d_2, ..., d_q\}$ includes the variable counterparts of the monitoring modules decisions. Algorithm 1 formulates all possible minterms based on the different sensor fault scenarios considered. Each minterm is treated as a subset of the characteristic function which models the state of the system and a temporary ZDD $T$ is generated to represent that subset. At each iteration, $T$ is initialized with $Empty()$ and for every variable $v_i \in v$ which has the value of 1 in the corresponding fault scenario, $v_i$ is inverted in $T$ using the $Change()$ function. Once all variables are set to the desired value $T$ is included in the fault dictionary using the $Union()$ function.

For example take a look at Figure 4 which demonstrates the generation of the highlighted minterm in Table III. Figure 4a
Algorithm 1 Enumerative Generation

Inputs: sensors set $S$, local sets $LS$, fault model $FM$
Output: ZDD-based fault dictionary $FD$

procedure $FDG(S, LS, FM)$
1: $\forall FS_i \in FM$ do
2: $v \leftarrow \text{minterm}(FS_i, LS)$
3: $T \leftarrow \emptyset$
4: $\forall v_i \in v$ do
5: \textbf{if} $v_i = 1$ \textbf{then}
6: $T \leftarrow !T(v_i)$
7: $FD = FD \cup T$
8: return $FD$

The Non-Enumerative generation is realised using cross-product operations. For the correct execution of this process, two variables per sensor and monitoring module were introduced. $s^0_q$ $(d^0_q)$ and $s^1_q$ $(d^1_q)$ for sensor (decision) 0 and 1 values. The two variable counterparts are mutually exclusive and can not have same value simultaneously.

Algorithm 2 describes the non-enumerative generation process and for simplicity reasons when the $i$-th sensor variable is set to 1, it implies $s^1_i = 1$ and $s^0_i = 0$ (same for decision variables).

The first step in the non-enumerative generation is to formulate the Characteristic Function (CF) which will include all fault scenarios considered by the fault model $FM$. As shown in the procedure Sensors CF this is done by calculating the cross-products between the sensor ZDDs. This procedure needs $n-1$ cross-product operations to complete, where $n$ is the number of sensors.

Next we generate the Local Fault Dictionaries, shown in the procedure Local FD. Monitor sensitization (ms), referred to at line 8, returns a set of variables which only includes the current sensor and all the decision variables it sensitizes (all other variables are don’t cares). Now, we can calculate the

---

Fig. 4. Formulation of the highlighted minterm in Table III. (a)Empty Set representation. (b)Inclusion of variable $D_1$, the $1^{\text{st}}$ with value $1$. (c)$s_1$, the $2^{\text{nd}}$ and last variable with value $1$ is added to the ZDD.

Fig. 5. ZDD generated for the system in Figure 2. Highlighted nodes form the traced path for $D(t) = [0, 0, 0, 1]$ which suggests $s_1$ is faulty.

---

shows the initialization point where $T$ represents the empty set, meaning that all variable values at this moment are 0. When variable $d_1$ is found that actually has the value 1 it is inverted and appears in $T$ as shown in Figure 4b (lines 6-8 of Algorithm 1). Figure 4c shows the completed ZDD for this minterm which only includes $d_1$ and $s_1$ since the rest of the variables are 0 and this is expressed with their absence. $T$ is then inserted in the final ZDD $F$, which represents the fault dictionary. This process is repeated until all possible fault scenarios are covered and the final fault dictionary form will be

$$FD = \bigcup_{i=1}^{f_{s_{\text{tot}}}} T_i$$

where $f_{s_{\text{tot}}}$ is the total number of possible sensor fault scenarios and $T_i$ are the respective minterms for each scenario.
Algorithm 2 Non-Enumerative Fault Dictionary Generation

Inputs: sensors set $S$, local sets $LS$, fault model $FM$
Output: ZDD-based fault dictionary $FD$

procedure Sensors Characteristic Function($S$)
generate $CF$ with the sensor fault combinations
1: $CF \leftarrow s_0$
2: $i = 1$
3: for $s_i \in S$ do
4: \hspace{1em} $CF \leftarrow CF \times s_i$
5: return $CF$

procedure Local FD($CF, LS$)
generate temporary fault dictionaries for each sensor
6: for $s_i \in S$ do
7: \hspace{1em} $LFD_i \leftarrow CF$
8: \hspace{1em} $ms_i \leftarrow$ monitor_sensitization($s_i, LS$)
9: \hspace{1em} $LFD_i \leftarrow LFD_i \times ms_i$
10: return $LFD[]$

procedure $FDG(LFD[j])$
generation of the Fault Dictionary with a series of cross product operations between the elements of $LFD$
11: $FD \leftarrow LFD_0$
12: for $s_i \in S$ do
13: \hspace{1em} $T \leftarrow$ SetDontCares($FD, LFD_i$)
14: \hspace{1em} $T \leftarrow T \times LFD_i$
15: \hspace{1em} $FD \leftarrow FD \cup T$
16: \hspace{1em} $T \leftarrow LFD_i \setminus T$
17: \hspace{1em} $T \leftarrow$ SetDVars($T, s_i$)
18: \hspace{1em} $FD \leftarrow FD \cup T$
19: return $FD$

function SetDontCares($FD, LFD$)
20: $FD_{dc} \leftarrow FD$
21: for $s_i \in FD_{dc}$ do
22: \hspace{1em} if $LFD(d_i) == 1$ then
23: \hspace{2em} $FD_{dc} \leftarrow \neg (FD_{dc} \cdot d_i)$
24: \hspace{1em} $FD_{dc} \leftarrow FD_{dc} \cup FD$
25: return $FD_{dc}$

function SetDVars($T, i$)
26: for $s_j \in S$, where $j < i$ do
27: \hspace{1em} for $d_i \in T$ do
28: \hspace{2em} if $ms_j(d_i) == 1$ then
29: \hspace{3em} $T(d_i) \leftarrow 0$
30: \hspace{1em} $T \leftarrow T \cup \text{temp}_{dc}$
31: return $T$

cross-product between the CF and each $ms_i$ which will return only the combinations from CF that include the current sensor along with the decision variables it affects. This is repeated for all sensors and need $n$ cross-product operations, where $n$ is the number of sensors.

Lastly, we use the resulting LFDs to build the final Fault Dictionary (FD). Procedure $FDG$ shows the steps required to generate the FD which will include all the fault signatures that may be produced by the scenarios considered in a given fault model. The procedure mostly consists of the basic operations shown in table II and the rationale is that we calculate the cross product between the LFDs to generate all the combinations. Some additional functions, SetDontCares() and SetDVars(), are introduced to maintain the correctness of the fault signatures.

We start this process by initializing the FD with the first LFD. Then, for each successive LFD, we have to set some necessary don’t care values (lines 13,20-25), in previously included minterms of the FD. These are all the decision variables sensitized by the respective sensor of the current LFD, and it is to ensure that all fault signatures will be updated according to the latest information. After we calculate the cross-product between previous history (with don’t cares) and the current LFD, the resulting minterms are included in the FD. The result of this operation will only return minterms with fault scenarios that have already been introduced in the FD earlier.

The next step is to extract the remaining fault scenarios from the LFD. At line 16, we calculate the set difference between the LFD and the result of the cross-product which will return the new fault scenarios. For these we know that none of the previously included sensors sensitizes the decision variables (otherwise they would have been in the result of the cross-product) so we can set them to 0 (lines 17,26-31). Final step, is to include the resulting minterms in the FD and repeat for all remaining LFDs.

IV. FAULT ISOLATION UNDER THE ZDD-BASED FAULT DICTIONARY

In the FDI scheme we consider, the fault isolation process is executed at the Aggregation module. This process is initiated every time a fault has been detected and a fault detection is indicated by the monitoring module decision values. Fault isolation is done based on the observed sensor fault pattern. Using the fault pattern (monitoring module decisions), we can extract all paths from the fault dictionary that include this pattern by co-factoring the ZDD based on the current decision variables. Co-factoring is done recursively using the SubSet() functions.

Let $D$ be the variable set which includes all the decision variables, by setting $F_{(d_i=x)}$ for all variables in $D$ the resulting ZDD should only contain paths with the sensor variables that could have produced the said pattern.

Assume the reference example system of Figure 2b. Consider the fault scenario $FS_1$ where $s_1$ is the only faulty sensor. The observed fault pattern then should be $D = \{0,0,0,1\}$. The highlighted path in Figure 5 shows the path formed from the fault pattern. Figure 6a shows what will be returned after performing the co-factoring. As expected, the result suggests that $s_1$ is the faulty sensor.

A case with multiple resulting paths is shown in Figure 6b where the fault pattern is $[0,1,1,0]$. This pattern could be produced by two different faulty sensors; $FS_3$ and $FS_7$. The resulting ZDD suggests sensors $s_2$ and $s_3$ as possibly faulty.
While $s_3$ is faulty in both scenarios, $s_2$ is only faulty in $F S_7$. This renders $s_3$ essential which means the produced ZDD can not have a path leading to terminal node ‘1’ without including $s_3$. Information about the essential variables of a ZDD can be gathered using the `Essential()` function. In this case $s_3$ belongs in the essential list and we can infer that it is faulty.

Even though $s_2$ is not in the essential list, we can not be sure if it’s healthy or not. Such cases, could have been solved if the system had a different configuration, but this is beyond the scope of the current work. Our future work will be focusing towards this end and try to improve a system’s overall isolability resolution.

V. EXPERIMENTS

A. Experimental Setup

The proposed method was implemented in C/C++ language using CUDD 3.0.0, the decision diagram package of [15]. Experiments were executed on a workstation server with a 2.5GHz Xeon E5-2670v2 and 94GB of system memory.

B. Dictionary Memory Requirements

We first evaluate the proposed approaches with respect to memory requirements. We compare the memory requirements for the two ZDD-based approaches presented in this work and the conventional approach of a 2D-array Signature Matrix, as used in [6].

Figure 7 reports the size of the fault dictionary for the different systems, when built with the three aforementioned approaches. Systems 1-4, as shown in the plot, consist of 50 sensors each and 5, 10, 25 and 50 local sets, respectively. For this experiment, up to 5 multiple faults are considered by the underlying fault model. Note that the y-axis is in logarithmic scale. The first observation is that both of the ZDD-based approaches reduce the size (in node count), by several orders of magnitude when compared to the conventional matrix-based representation. Secondly, as expected the non-enumerative approach needs approximately twice the memory of that of the enumerative due to the double amount of variables required to build it. Still, this size is considerably smaller than that of the matrix approach. We note that each node in a ZDD requires a small number of bytes to be represented, hence, the overall storage requirements for the generated fault dictionaries are in the order of few MBs (depending on the system size with respect to the number of sensors and sensor sets).

C. ZDD Dictionary Generation Time

Even though the non-enumerative approach has a bigger memory footprint than the enumerative approach, it has a huge advantage in generation time. This is shown in Figure 8 which plots the time required to generate the dictionaries of the previous experiment, for the proposed approaches. There is a significant improvement of several orders of magnitude in generation time and this can help the designers of a system to test out multiple alternative configurations with minimal time expenses. Testing different configurations of a system is important for exploring aspects like the isolability resolution and finding an optimized solution for the needs of the system. Further exploration towards this end is considered for our future work.

D. Fault Isolation Time

One of the major benefits of ZDDs, is the fast access time. This allows for the fault isolation process to happen in time linear to the size of the dictionary, as it requires a small number of linear traversals on the ZDD (as discussed in Section IV).

Figure 10 shows the average time spent to isolate faults in different systems. The systems considered here consist of 24 sensors and 4, 6, 8 and 12 local sensor sets. The secondary axis (bars) show the size for each system’s dictionary.
Our results report an average of around 0.012s for the system with 12 local sensor sets, while going all the way down to 0.002s for the system with 4 local sensor sets.

VI. CONCLUSIONS

With this work we provide an efficient and scalable solution for performing sensor fault isolation in IoT enabled cyber-physical systems. We propose the use of ZDDs for building the Fault Dictionary. The experiments performed demonstrate that the size of the dictionary scales linearly for systems with large number of sensors and small number of local sensor sets. The size remains reasonable even for larger numbers of local sets and multiple fault cardinalities.

Both of the proposed ZDD generation approaches outperform the conventional 2D-array approach by several orders of magnitude in terms of memory footprint. Furthermore, we demonstrate how fast fault isolation process can be achieved when using the ZDD-based fault dictionary, allowing for real-time response by the aggregator of the monitoring and control module of the system.

For our future work, we consider using the non-enumerative generation approach for further exploration of a system’s properties with respect to FDI. The low generation time of the fault dictionary enables testing of alternative configurations of a given system without significant time expenses. This can lead to optimized configurations with improved overall diagnostic performance.

ACKNOWLEDGMENT

This work has been supported by the EU Horizon 2020 research and innovation programme under grant agreement No 739551 (KIOS CoE) and from the Government of the Republic of Cyprus through the Directorate General for European Programmes, Coordination and Development.

REFERENCES

[1] Antsaklis, Panos J., et al. "Control of cyberphysical systems using passivity and dissipativity based methods." European Journal of Control 19.5 (2013): 379-388.
[2] Kröger, W., Zio E., and M. Schlüper. "Vulnerable systems." Springer Verlag, 2011.
[3] Isermann, Rolf. "Fault-diagnosis systems: an introduction from fault detection to fault tolerance." Springer Science & Business Media, 2006.
[4] Mhamdi, Lotfi, et al. "Multiple fault diagnosis using mathematical models." 2013 9th Asian Control Conference (ASCC). IEEE, 2013.
[5] Fragkoulis, D., G. Roux, and B. Dahhou. "A global scheme for multiple and simultaneous faults in system actuators and sensors." 2009 6th International Multi-Conference on Systems, Signals and Devices. IEEE, 2009.
[6] Reppa, Vasso, Marios M. Polycarpou, and Christos G. Panayiotou. "Sensor fault diagnosis," Foundations and Trends® in Systems and Control 3.1-2 (2016): 1-248.
[7] S. Minato, "Zero-Suppressed BDDs for Set Manipulation in Combinatorial Problems," 30th ACM/IEEE Design Automation Conference, Dallas, TX, USA, 1993, pp. 272-277.
[8] Bryant, Randal E. "Graph-based algorithms for boolean function manipulation." Computers, IEEE Transactions on 100.8 (1986): 677-691.
[9] K. S. Brace, R. L. Rudell and R. E. Bryant, “Efficient implementation of ZDDs”. In Proceedings of the 27th ACM/IEEE Design Automation Conference, 1990.
[10] Padmanaban, Saravanan, Maria K. Michael, and Spyros Tragoudas. "Exact path delay fault coverage with fundamental ZBDD operations." IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems 22.3 (2003): 305-316.
[11] Kocan, Fatih, L. Li, and Daniel G. Saab. "Exact path delay fault coverage calculation of partitioned circuits." IEEE Transactions on Computers 58.6 (2008): 858-864.
[12] Neophytou, Stelios N., and Maria K. Michael. “Optimal variable ordering in zbdd-based path representations for directed acyclic graphs.” 2014 IEEE 32nd International Conference on Computer Design (ICCD). IEEE, 2014.
[13] Yoon, Sungroh, and Giovanni De Micheli. "An application of zero-suppressed binary decision diagrams to clustering analysis of DNA microarray data." The 26th Annual International Conference of the IEEE Engineering in Medicine and Biology Society. Vol. 2. IEEE, 2004.
[14] Kurai, Ryutaro, Shin-ichi Minato, and Thomas Zeugmann. "N-gram analysis based on zero-suppressed BDDs." Annual Conference of the Japanese Society for Artificial Intelligence. Springer, Berlin, Heidelberg, 2006.
[15] Somenzi, Fabio. "CUDD: CU decision diagram package." http://vlsi.colorado.edu/~fabio/CUDD/ (1997).
[16] Alippi, Cesare, Stavros Ntalampiras, and Manuel Roveri. "Model-free fault detection and isolation in large-scale cyber-physical systems." IEEE Transactions on Emerging Topics in Computational Intelligence 1.1 (2016): 61-71.
[17] Maurya, Mano Ram, Raghunathan Rengaswamy, and Venkat Venkata- subramanian. "Fault diagnosis by qualitative trend analysis of the principal components." Chemical Engineering Research and Design 83.9 (2005): 1122-1132.
[18] Kocian, Jan Maciej, Michal A Barty, and Michal A Syfert. "Method of multiple fault isolation in large scale systems." IEEE Transactions on Control Systems Technology 20.5 (2011): 1302-1310.