Testing Deterministic Avionics Networks Using Orthogonal Arrays

Muhammed Efe Altuntaş[1,2], Muhammed Şeker[1,2]
Pınar Kışla[1,2], Eyüp Can Akpolat[2]

[1]*Yildiz Technical University*
Dept. of Avionics Engineering
Istanbul, Turkey
[2]*Turkish Aerospace Industries, Inc.*
Ankara, Turkey
email: muhammedefe.altuntas@tai.com.tr

İbrahim Hökelek
Muhammet Selim Demir

TÜBİTAK, BİLGEM
Kocaeli, Turkey
email: ibrahim.hokelek@tubitak.gov.tr

Erhan Akdoğan

[1]*Yildiz Technical University*
Dept. of Mechatronics Engineering
[2]*Health Institutes of Turkey*
İstanbul, Turkey
email: eakdogan@yildiz.edu.tr

***Abstract*—ARINC 664 is a widely used Ethernet-based Deterministic Network (DTN) standard providing the bounded latency and jitter for safety critical applications in the avionics systems. It is not possible to generate a test suite with a feasible size without employing a formal method such as Orthogonal Arrays (OAs) if we consider all possible combinations of DTN switch configuration parameters such as the number of Virtual Links (VLs), the frame size, and bandwidth allocation gap for each connection. However, using OAs is not sufficient to generate test cases representing real-life usage of DTN switches since many combinations of VL parameters may be inadvertently eliminated by the OA generation process. In this paper, we present an approach, called *chained* OA*s*, where multiple applications of OAs are applied to the test factors to generate test cases that are representative of real-life usage of DTN switches. The test cases are converted into executable test scripts to be run by the AIM commercial test system. This study shows that chained OAs method is capable of reducing prohibitively large number of tests required for complex systems, while covering pair-wise combinations of all test parameters.**

***Index Terms*—orthogonal array; test generation; deterministic network; ARINC 664; avionics**

I. INTRODUCTION

Ethernet-based Deterministic Network (DTN) technologies have been developed to cope with the high speed data communication requirements of the modern avionics applications. ARINC 664, which is a commonly used DTN technology, provides the bounded latency and jitter when safety critical applications communicate with each other over the avionics networks within an aircraft. Because of their safety-critical nature, the availability of large number of configuration options and high speed operation using thousands of virtual links, testing ARINC 664 switches is a challenging task.

Prohibitively great numbers of combinations of DTN configurations necessitates that formal test generation techniques be employed to effectively reduce the number of tests. Orthogonal Array (OA) is a popular technique to optimize test suite size for systems with many features while maintaining their coverage [1]–[7]. However, using a single level OAs is not sufficient to generate test cases representing realistic operating conditions of DTN switches. Many combinations of switch configuration parameters may be inadvertently eliminated by the OA optimization process. In this paper, we present an approach, called *chained* OA*s*, where multiple and consecutive applications of OAs are employed to generate tests that are representative of real-life usage of DTN switches. Chained OAs method allows for preserving the number of combinations for a set of selected features, while eliminating redundant combinations for others as decided by test engineers. We define two stages for the chained OA method in test generation of DTN switches. First, combinations of different number of VLs that can be assigned to each switch port are generated by OAs. Then, the OA output elements are used as inputs to a set of second stage OAs to determine the remaining switch parameters such as transmission time gaps between consecutive frames, minimum and maximum frame sizes, and VL priorities. The OA outputs are generated using an open source Python library, called *oapackage* [8]. The tests generated by the chained OAs method are converted into executable scripts, which are applied to a DTN switch under test by a commercial test system, called AIM [9].

This paper is organized as follows. ARINC 664 avionics deterministic networks are outlined in Section II. The fundamental features of OAs are introduced in Section III. Section IV presents the complexity of test space for avionics deterministic networks and impact of the chained OA method to reduce the test suite size while preserving important test features. The architecture of our DTN switch test system is summarized in Section V.

II. ARINC 664 - AVIONICS DETERMINISTIC NETWORK

The most important elements of an ARINC 664 DTN include Switch (SW), End Systems (ESs), and Virtual Links (VLs) [10]. Applications running on different ESs communicate with each other using VLs that are routed by SW. The bounded latency and jitter requirements for VLs are met by offline planning of ARINC 664: the number of VLs, the source and destination ESs, the minimum and maximum frame sizes, and Bandwidth

 ISBN: 978-1-61208-886-0

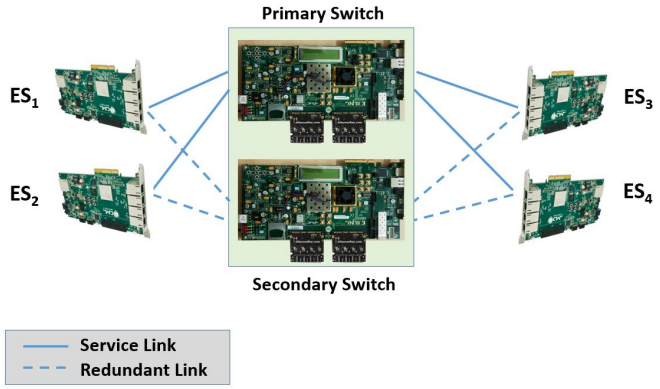

Fig. 1. An example ARINC 664 DTN with 2 SWs and 4 ESs

Allocation Gap (BAG) for all VLs [11]–[13]. The main idea behind providing deterministic communication is to limit the amount of traffic injected by each VL to the network by setting allowed frame size and BAG values, to achieve the desired delay and jitter objectives. BAG specifies the minimum transmission time between the first bits of two consecutive frames. A traffic shaping mechanism running over each VL will ensure that the amount of traffic injected by each node will be controlled and the traffic exceeding the average and peak rates will be shaped. The switch will enforce filtering and policing requirements, limiting fault propagation, and ensuring determinism.

In the ARINC 664 redundancy management architecture, the same frame is transmitted over two independent SWs and one of the received copies is sent to the application layer at the receiver. ARINC 664 does not provide a reliable transmission guarantee that all frames will be delivered to their destinations. However, in a properly planned network, the amount of congestion can be predicted and limited during the planning phase and the priority concept along with the redundancy management provide a reliable data transmission service. For the safety certification of ARINC 664, it is a common practice that the Network Calculus (NC) framework can be applied to calculate the worst-case delay and jitter bounds for a given network configuration [14] [15]. Figure 1 depicts an ARINC 664 DTN with two SWs, one primary and the other secondary, and four ESs, each of which is connected to the SWs via service and redundant VLs.

III. Orthogonal Arrays

Orthogonal Arrays (OAs) [7] are among mathematical tools to design an optimum mix of multiple variables in a set of experiments. They are effective in reducing the number of experiments, where, otherwise, a prohibitively large number of combinations would be needed [1]–[3] [7]. They are classified as the same and mixed level OAs as described in Sections III-A and III-B, respectively.

A. Same level OAs

In general, an OA is defined as a 4-tuple:

$$OA(N, F, L, S), \tag{1}$$

where

- N is the number of experiments (also referred to as *runs*) which represents the number of rows in an OA.
- F, called a *factor*, is the number of columns in an OA, where each column corresponds to a variable of an experiment.
- L is the number of *levels*, representing the number of distinct values that a factor can take. It is possible that, in a given experiment, different factors may possess different levels.
- S is called the *strength* of an OA ($0 < S \leq F$). An OA is designed in such a way that the number of rows needed to cover S-wise combinations of variables is given as L^S [1]. When $S = F$, all possible combinations of variables will be included in the experiment. A user can select the degree of combinations of variables to be considered by setting smaller values for S.

When all F factors have the same number of L levels, the OAs are referred to as the *same level* OAs. For a given experiment having F factors with L levels, there are L^F possible runs to cover all combinations of variables (assuming that all factors have the same number of levels L). OAs allow for reducing the number of runs to L^S by selecting an S value to cover S-wise combinations of variables. For example, all pairwise combinations of variables can be covered when S is set to 2. The number of runs can be further reduced as $N = \lambda \cdot L^S$ due to time limitations, where λ is a real number between 0 and 1 [5] [16]. Selection of λ is determined by testers depending on the complexity and the number of parameters of a given system under test such that smaller values are preferred for complex systems.

Let us consider an experiment with $F = 3$ factors, each with $L = 2$ levels. Without using an OA, the number of runs required to cover all possible combinations of variables is 2^3 as shown in Table I (a). We can observe that all possible situations between any two factors appear more than once in Table I (a). For example, $(f_1 = 0, f_2 = 1)$ is listed twice for factors 1 and 2 in runs 3 and 4, respectively. Orthogonal arrays eliminate such repetitions and reduce the total number of runs for experiments with large number of factors and levels. If we design an OA for this example with all factors having 2 levels and selecting the strength as $S = 2$, we will have a total of $N = 2^2$ runs. In this case, the OA will be represented as $OA(4, 3, 2, 2)$ as shown in Table I (b).

B. Mixed level OAs

In the OAs described in Section III-A, all factors have the same number of levels. However, it is possible that, in real-life situations, some factors may have different number of levels to be tested. For example, in an ARINC 664 switch, mapping incoming traffic flows to outgoing ports may have a large number of possible assignments to be tested (e.g., up to 24 ports), whereas the number of different priorities for a flow

TABLE I. EXAMPLE FOR SAME LEVEL OA: (a) AN EXPERIMENT WITH 3 FACTORS EACH WITH 2 LEVELS AND (b) RUNS FOR $OA(4, 3, 2, 2)$

Runs	f_1	f_2	f_3
1	0	0	0
2	0	0	1
3	0	1	0
4	0	1	1
5	1	0	0
6	1	0	1
7	1	1	0
8	1	1	1

(a)

Runs	f_1	f_2	f_3
1	0	0	0
2	0	1	1
3	1	0	1
4	1	1	0

(b)

TABLE II. EXAMPLE RUNS FOR A MIXED LEVEL $OA(12, 2^4 3^1, 2)$

Runs	f_1	f_2	f_3	f_4	f_5
1	0	0	0	0	0
2	0	1	0	1	0
3	1	0	1	0	0
4	1	1	1	1	0
5	0	0	1	1	1
6	0	1	1	0	1
7	1	0	0	0	1
8	1	1	0	1	1
9	0	0	1	1	2
10	0	1	0	0	2
11	1	0	0	1	2
12	1	1	1	0	2

may require much fewer options (e.g., two priorities of *High* and *Low*). OAs whose factors have different levels are called the *mixed level* OA*s*. The tuple defining a mixed level OA is given as [1]:

$$OA(N, L_1^{|F_1|} L_2^{|F_2|} \dots L_v^{|F_v|}, S) \tag{2}$$

where $|F_i|$ is the number of different factors with the same number of levels L_i for $(i = 1, 2, \dots, v)$ and $\sum_{i=1}^{v} |F_i| = F$. N and S are defined in Eq. (1). For a factor F_i with the number of levels L_i, the number of possible runs will be ${L_i}^{|F_i|}$, which may be restricted by selecting an appropriate value of strength S as L_i^S as discussed above for the same level OAs. N, the number of runs, is selected in such a way that it is the least common multiplier of the number of $S-$wise factor level combinations. For a pairwise coverage, the combinations between any two factors can be generated as $L_i \cdot L_j$ for $i \neq j$ and $i, j = 1, \dots, F$. The generalization to $S-$wise combinations is studied in [1].

As an example, consider an experiment with $F = 5$ factors, where $F_1 = 4$ of them are with $L_1 = 2$ levels and one factor $F_2 = 1$ with $L_2 = 3$ levels. For the factors having 2 levels, the number of possible runs is $L_1^{|F_1|} = 2^4$, whereas, for the factor having 3 levels, there are $L_2^{|F_2|} = 3^1$ possibilities. Without using OAs, there are a total of $L_1^{|F_1|} \cdot L_2^{|F_2|} = 48$ runs for this example experiment. Suppose that we are interested in pair-wise combinations of these factors (i.e., $S = 2$). A mixed level OA can be defined as $(12, 2^4 3^1, 2)$, where $N = 12$ is calculated as the least common multiplier between the pair-wise combinations of any two factors. Since there are two levels (i.e., 2 and 3), the number of different combinations between any two factors can only be $2 \cdot 2 = 4$ or $2 \cdot 3 = 6$ with the least common multiplier is 12, which is the number of runs in Table II.

IV. TESTING AVIONICS NETWORKS USING ORTHOGONAL ARRAYS

A. Test Space for DTN Switch

While traditional Ethernet switches perform switching function by routing frames from input ports to output ports by means of MAC tables, DTN switches utilize statically configured VL tables to route frames from input to output ports, where each frame carries VL identifier within its frame header to be used for switching purpose. An example DTN switch configuration, which is selected as the System Under Test (SUT), uses 4 ports and operates at the speed of 1 Gbps. The DTN switch includes filtering, policing, and switching features. A selected subset of parameters defining these features such as minimum and maximum frame lengths, BAG, and priority are shown in Table III.

For this example switch configuration, there are eight different BAG values (in milliseconds) each indicating the time slot in which a VL needs to be transmitted. For example, VLs with BAG = 2ms are guaranteed to be transmitted within an interval of 2ms. L_{min} and L_{max} values define the minimum and maximum frame sizes, respectively ($L_{min} < L_{max}$). Although the ARINC 664 standard allows for frames between 64 and 1518 bytes [11], different minimum and maximum frame sizes within these limits are permitted. For example, if a VL selects $L_{min} = 100$ bytes, the switch will not permit a 99-byte packet to be transmitted in this VL. A small sample of L_{min} and L_{max} values are shown in Table III. Determining the BAG values for all VLs, together with L_{min} and L_{max} values, will specify the maximum speed of operation for a switch. In deterministic networks, it is important to classify packets by their importance so that they are prioritized in switch queues. For this example configuration, priority values are defined as *Low* and *High*. The number of incoming and outgoing VLs into a given port are selected to be eight distinct values between 192 and 3840.

B. Complexity of the Test Space

The number of VLs is an important parameter possibly indicating the traffic utilization rate of the ports and eventually the determinism features of a switch. Based on Table III, there are $4 \cdot 8 \cdot 8 \cdot 8 \cdot 2 \cdot 8 \cdot 8 = 262{,}144$ different configurations for one VL. Since the maximum number of VLs in our example SUT implementation is 4096, the number of possible combinations of different VL configurations is prohibitively large. In an example configuration driven from Table III, port 2 sends 1152 VLs to output ports 3 and 4 with BAG = 64, $L_{min} = 128$, $L_{max} = 1518$ and with high priority, whereas port 3 receives 384 VLs and port 4 the remaining 768 VLs (the total number of VLs sent and received will be equal among all ports). For the parameters shown in Table III, the total number of combinations C for a given VL becomes

$$C = B^V \cdot L_s^V \cdot L_u^V \cdot p_i^V \cdot P \tag{3}$$

TABLE III. SAMPLE CONFIGURATION PARAMETERS FOR DTN SWITCH

Number of Ports	BAG (ms)	L_{min} (Bytes)	L_{max} (Bytes)	Priority	Incoming VLs to a Port	Outgoing VLs from a Port
1	1	64	1518	High	192	192
2	2	128	1400	Low	384	384
3	4	150	1300		576	576
4	8	200	1200		960	960
	16	300	1100		1152	1152
	32	400	1000		1920	1920
	64	512	900		2880	2880
	128	750	800		3840	3840

where B, V, L_s, L_u, p_i and P are the number of different values for BAG, VLs, L_{min}, L_{max}, priorities, and the number of ports, respectively. As an example with 1152 VLs for each port, the resulting complexity of the test space can be expressed using Eq. (3) as:

$$C = 8^{1152} \cdot 8^{1152} \cdot 8^{1152} \cdot 2^{1152} \cdot 4 \tag{4}$$

It is clear from Eq. (4) that it is not possible to generate a test suite with a feasible size without employing formal methods such as OAs. However, using OAs given in Table III may not generate test cases representing real-life usage of DTN switches. For example, a run generated from Table III with 2 ports, BAG = 4ms, $L_{min} = 300$, $L_{max} = 1000$, and high priority implies that all VLs will have the same number of BAG, L_{min} L_{max} and priority values. But, in a realistic scenario, each VL can independently have any of the parameter values given in Table III. In Section IV-C, we present an approach, where a sequence of OAs are employed to remedy this limitation.

C. Using Chained OAs for DTN Test System

As described in Section IV-B, the DTN network features define a prohibitively large test space. Although OAs can be used to reduce the test space significantly by limiting the coverage to a subspace with pairwise combinations of factors, directly using the parameters in Table III may inadvertently eliminate important test cases representing real-life usage of DTN switches. In this paper, we use *chained* OA*s*, where the test factors are divided into groups and OAs are generated sequentially such that the outputs of the first OA is used as the inputs for the next OA. Without loss of generality, let us first present the chained OA for two groups of input parameters, namely F_{g1} and F_{g2}, where $F_{g1} \cup F_{g2} = F$. With chained OAs, first, the parameters for F_{g1} that are considered to be more influential in system performance are selected and OA runs are generated with possibly higher values of strength S_{g1}. This way, suppression of different combinations is kept at minimum for this subset. After F_{g1} is handled, subsequent OAs are generated using the remaining parameters.

Let us consider an OA, called OA_{g1}, with N_{g1} runs, F_{g1} factors, each with level L_{g1}, and strength of S_{g1}:

$$OA_{g1}(N_{g1}, L_{g1}^{F_{g1}}, S_{g1})$$

OA_{g1} generates an orthogonal array with N_{g1} rows and F_{g1} columns as follows:

TABLE IV. DTN SWITCH VARIABLES FOR OAs: (a) THE NUMBER OF INCOMING VLs TO PORTS FOR OA_{g1}, AND (b) TEST PARAMETERS FOR VLs FOR $OA_{i,j}$

Number of VLs			
Port 1	Port 2	Port 3	Port 4
192	192	192	192
384	384	384	384
576	576	576	576
960	960	960	960
1152	1152	1152	1152
1920	1920	1920	1920
2880	2880	2880	2880
3840	3840	3840	3840

(a)

Number of Ports	BAG (ms)	L_{min} (Bytes)	L_{max} (Bytes)	Priority
1	1	64	1518	High
2	2	128	1400	Low
3	4	150	1300	
4	8	200	1200	
	16	300	1100	
	32	400	1000	
	64	512	900	
	128	750	800	

(b)

$$\begin{matrix} n_{1,1} & n_{1,2} & \dots & n_{1,F_{g1}} \\ n_{2,1} & n_{2,2} & \dots & n_{2,F_{g1}} \\ \vdots & \vdots & \vdots & \vdots \\ n_{N_{g1},1} & n_{N_{g1},2} & \dots & n_{N_{g1},F_{g1}} \end{matrix}$$

where $n_{i,j}$ is the element of run i for factor j. Using the chained OAs approach, for each element $n_{i,j}$ of OA_{g1}, we generate a secondary OA, called $OA_{i,j}$:

$$OA_{i,j}(n_{i,j}, L_{g2}^{F_{g2}}, S_{g2})$$

where F_{g2}, L_{g2} and S_{g2} are the number of factors, levels and the strength selected for the secondary level orthogonal arrays, respectively ($S_{g2} \leq S_{g1}$). After applying the chained OAs, the total number of runs will be:

$$\sum_{i=1}^{N} \sum_{j=1}^{F_{g1}} (OA_{i,j}) \tag{5}$$

where $OA_{i,j}$ generates a matrix with dimensions of $(n_{i,j} \times F_{g2})$. For the general case with R subsets of F factors, where $F_{g1} \cup \dots \cup F_{gR} = F$, subsequent OAs can be generated in a similar manner.

In testing a DTN switch, the number of VLs is the most critical parameter since it directly impacts the determinism of traffic flows through port utilization. Therefore, the first OA, called OA_{g1}, is defined to generate runs with different number of VLs for each of the four incoming and outgoing ports. Once these runs are generated by OA_{g1}, another OA, called OA_{g2}, is needed to cover the individual parameters of each of these VLs.

In defining OA_{g1}, the range of 1 to 4096 possible VLs for each of the four ports of a switch is arranged for this

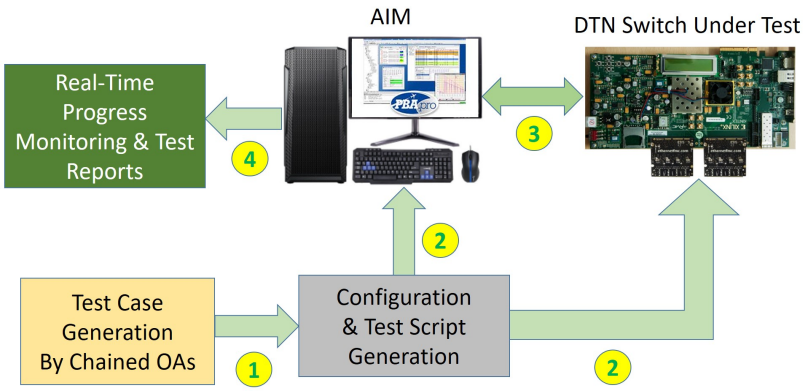

Fig. 2. Architecture and information flow of DTN test system

study by eight different values from 192 to 3840 as shown in Table IV(a): $OA_{g1}(256, 8^4, 2)$, where 256 is selected as the feasible number of runs and the strength is chosen as $S = 2$. Note that since the switch supports a maximum of 4096 VLs, any cases that exceed this upper limit in OA_{g1} are eliminated, resulting in 256 rows being reduced to 143.

VL configurations assigned by OA_{g1} to each switch port are determined by $OA_{i,j}$ using the five (5) parameters shown in Table IV(b). These parameters are selected as the most influential ones on testing switch performance: $OA_{i,j}(n_{i,j}, 4^1 8^3 2^1, 2)$, where $n_{i,j}$ is the number of runs for element at i^{th} row and j^{th} column generated by OA_{g1}.

For the sake of simplicity, this study assumes that there are 4 ports in this implementation of a DTN switch and, hence, there are a total of $143 \cdot 4 = 572$ orthogonal arrays generated based on Table IV(a):

$$\sum_{i=1}^{143} \sum_{j=1}^{4} OA_{i,j} \tag{6}$$

where $OA_{i,j}$ is a matrix with the dimensions of $(n_{i,j} \times F_{g2})$, $n_{i,j}$ ranging from 192 to 3840 from Table IV(a) and $F_{g2} = 5$ is the number of columns in Table IV(b). Therefore, 572 different matrices with the dimensions of $n_{i,j} \times 5$ are generated for this simplified version of DTN switch, where each matrix corresponds to a different switch configuration. The total number of VL configurations ranges approximately from $143 \cdot 4 \cdot 192 =$ 109,824 to $143 \cdot 4 \cdot 960 =$ 549,120. Note that there are multiple test cases in each configuration as described in Section V.

V. Implementation of DTN Test System

A. System Architecture

High level architecture of a DTN switch test system and its information flow are presented in Figure 2. First, OA model parameters representing realistic operation of the switch are determined as presented in Tables III, IV(a) and IV(b). These parameters include the number of switch ports, possible number of VL combinations to be used for each port and the set of VL parameters to be included in OA array runs. Selection of switch parameters is an important step in test generation process since there are typically multiple trade-offs between a feasible number of test cases and a desired coverage of switch features that need to be decided together by the systems engineers, developers and test engineers. The current implementation of the DTN test system generates executable test scripts using the chained OAs method described in Section IV-C, which are applied to an SUT using a commercial tool called AIM [9].

Once the OA parameters are determined (e.g., the number of runs and the factors with their levels), the next step is to construct the OAs with the desired level of strength. In step 1 of Figure 2, the OA tables were generated using an open source Python library, called *oapackage* [8], whose capabilities were suitable for the study presented in this paper. The chained OA method uses two different stages of OAs for this case study. In the first OA output, combinations of different number of VLs that can be assigned to each of the four switch ports are generated. The inputs used in this paper for generating the first OA are given in Table IV(a). Then, each element of first OA output is used as the number of runs for the second stage OA to determine the remaining parameters such as BAG, L_{min}, L_{max}, and priority as shown in Table IV(b).

The outputs of the chained OA method are now ready to be converted to SUT configurations, which define the operational characteristics of a DTN switch. Since AIM is used as the tool to apply the tests to the SUT, similar conversions are performed to prepare the corresponding AIM configurations, as depicted in step 2 of Figure 2. With these configurations, test cases are executed by generating data frames, sending and receiving them to and from the SUT ports and checking their validity with respect to the expected outputs (step 3 in Figure 2).

Because there are large number of tests to be executed on SUT, logging the test results and generating appropriate reports are important for the test system (step 4 in Figure 2). In addition, since the tests are expected to run for many hours in a continuous manner, observing the status of the test progress in real-time is an important requirement for the test system. The GUI shown in Figure 3 is an example for test monitoring capabilities of the test system, which depicts switch configurations, the number of tests that are already run,

 ISBN: 978-1-61208-886-0

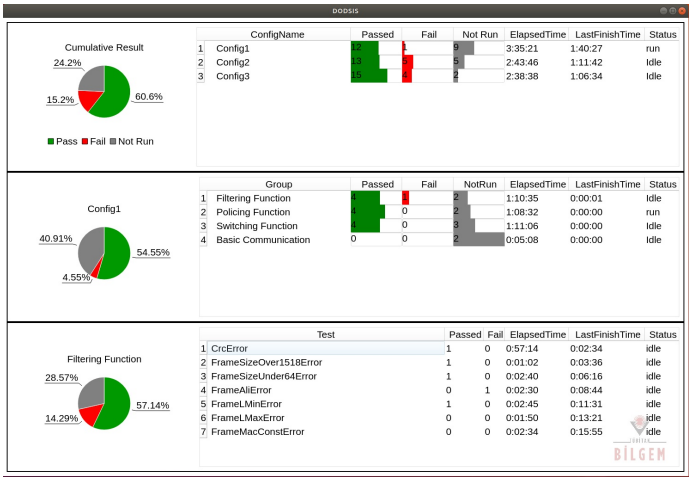

Fig. 3. GUI sample for real-time status monitoring of DTN test system

pass/failure ratios, and the expected time of the completion. By clicking on any of the tabs on the GUI, test engineer will have access to more detailed outcomes of a given test.

B. Current Status of Test System

We estimate that, using chained OAs, the test suite will deploy hundreds of thousands of VLs to verify real-life scenarios that the SUT is expected to be exposed. Currently, the test suite includes 143 switch configurations, each of which represents a different setting for distinct usage of the DTN switch in the field. Each switch configuration will be run continuously for 3-4 hours, representing a realistic load for typical avionics applications.

VI. Conclusions and Future Work

Test generation for network switches used in safety critical avionics systems is a significant challenge since these systems possess a large number of configuration options and high speed operation using thousands of virtual connections. There are prohibitively large number of test cases if one wishes to include all possible combinations of switch configuration parameters. One popular tool to reduce number of tests is OAs, especially suitable for complex industrial applications. In this paper, we presented chained OAs method, where test parameters are divided into multiple groups and the OA generated for a group is used as an input for the next group. Independent of implementation or employed test architecture, chained OAs method can be applied to any SUT with large number of test parameters. For test generation of DTN switches, we defined two groups, where the first group deals with different number of virtual connections to be assigned to each switch port and the second one determines the remaining switch parameters such as transmission time gaps, minimum and maximum frame sizes and priorities. The test cases generated by the chained OAs method were converted into executable test scripts, which were applied to an SUT by a commercial test system.

Future extension of this work includes development of OAs that incorporate parameters defined for single and multiple end systems and reliability test suites for interoperability of switch and end systems for avionics applications. We expect that these extensions will be capable of representing real-life usage of deterministic avionics network systems and, hence, provide more effective test cases. We also plan to present a formal study to explicitly quantify the complexity and efficiency trade-off achieved by the chained OAs approach.

References

[1] A. S. Hedayat, N. J. A. Sloane, and J. Stufken, *Orthogonal Arrays: Theory and Applications*. Springer, ISBN 978-1-4612-1478-6, 1999.

[2] M. S. Phadke, *Quality Engineering Using Robust Design*. Prentice-Hall, ISBN 0-13-745167-9, 1989.

[3] K. Gopalakrishnan, D. Stinson, and D. Cheriton, "Applications of Orthogonal Arrays to Computer Science," in *Proc. of ICDM.*, 2006, pp. 149 – 164.

[4] G. Taguchi and S. Konishi, *Taguchi Methods: Orthogonal Arrays and Linear Graphs - Tools for Quality Engineering*. American Supplier Institute, Center for Taguchi Methods, ISBN 978-0-94-1243-01-8, 1987.

[5] R. Kacker, E. Lagergren, and J. Filliben, "Taguchi's Orthogonal Arrays are Classical Designs of Experiments," *Journal of Research of the National Institute of Standards and Technology*, vol. 96, no. 5, p. 577, 1991.

[6] E. D. Schoen, P. T. Eendebak, and M. V. M. Nguyen, "Complete Enumeration of Pure-Level and Mixed-Level Orthogonal Arrays," *Journal of Combinatorial Designs*, vol. 18, no. 2, pp. 123–140, 2010, retrieved 15.05.2021. [Online]. Available: https://onlinelibrary.wiley.com/doi/abs/10.1002/jcd.20236

[7] S. Yazar, "Reconstruction of the Taguchi Orthogonal Arrays with the Support Vector Machines Method," *Balkan Journal of Electrical and Computer Engineering*, vol. 9, pp. 129 – 137, 2021.

[8] P. Eendebak, "The Orthogonal Array Package," retrieved 04.04.2021. [Online]. Available: http://www.pietereendebak.nl/oapackage/index.html

[9] "AFDX / ARINC 664P7 Modules with a Difference," retrieved 15.05.2021. [Online]. Available: https://www.aim-online.com/products-overview/modules/afdx-arinc664p7/

[10] J.-P. Moreaux, "Data transmission system for aircraft," U.S. Patent US6925088B1, Nov. 1999.

[11] *ARINC 664, P7: Avionics Full Duplex Switched Ethernet (AFDX) Network*. INC Aronautical Radio, 2005.

[12] E. C. Akpolat et al., "Genetic Algorithm Based ARINC 664 Mixed Criticality Optimization Using Network Calculus," in *2021 IEEE International Conference on Communications Workshops (ICC Workshops)*, 2021, pp. 1–6.

[13] M. N. Durceylan, O. F. Gemici, I. Hokelek, and H. Asmer, "Reconfigurable arinc 664 end system implementation on fpga," in *2020 28th Signal Processing and Communications Applications Conference (SIU)*, 2020, pp. 1–4.

[14] M. Yeniaydin et al., "Priority Re-assignment for Improving Schedulability and Mixed-Criticality of ARINC 664," in *2021 IFIP Networking Conference (IFIP Networking)*, 2021, pp. 1–6.

[15] A. Bouillard, M. Boyer, and E. Le Corronc, *Deterministic Network Calculus: From Theory to Practical Implementation*. John Wiley & Sons, ISBN 978-1-119-56341-9, 2018.

[16] H. Bayrak and A. Alhan, "On the Construction of 2-Symbol Orthogonal Arrays," *Hacettepe Journal of Mathematics and Statistics Volume*, vol. 31, pp. 57–62, 01 2002.

 ISBN: 978-1-61208-886-0