JOURNAL OF AEROSPACE INFORMATION SYSTEMS
Vol. 21, No. 5, May 2024

OMNeT++ Simulation Framework for Avionics Full-Duplex Switched Ethernet

İ. Peşkircioğlu Gökçe* and M. Üçüncü†
Başkent University, 06790 Ankara, Turkey
and
E. Güran Schmidt‡
Middle East Technical University, 06800 Ankara, Turkey

https://doi.org/10.2514/1.I011351

Avionics Full-Duplex Switched Ethernet (AFDX) is a popular network technology for integrated modular avionics (IMA). When designing an AFDX network, it is important to fulfill the real-time communication requirements of avionic systems. To this end, simulation tools offer a low-cost solution for the performance evaluation of a given AFDX network and message set configuration at the design stage. In this paper, we develop an improved AFDX model for the widely used OMNeT++ (extensible, modular, component-based C++ simulation library and framework) network simulator. Our model exhibits high fidelity to the standard, adds detailed parameter configuration and data logging capabilities, and is compatible with the most recent OMNeT++ version. Our model is accepted by the OMNeT++ community and published as the current AFDX simulation model. Moreover, we propose a complete AFDX simulation framework by augmenting our improved OMNeT++ models with a new network configuration and analysis software tool that we call ANCAT. ANCAT enables users who lack familiarity with the OMNeT++ interface and specific file formats to provide simulation parameters in generic file formats such as.xlsx. Subsequently, ANCAT runs the simulation and generates reports for the results. We first systematically verify the correctness of our AFDX simulation framework by custom-designed experiments. Then, we investigate the practicability of our proposed framework with the performance analysis of a realistic AFDX network topology and message set. In particular, we demonstrate that our proposed framework can be efficiently used to explore the real-time capabilities and relevant features of AFDX networks, as well as provide basic guidelines for network configuration.

Nomenclature

PSN	=	previous sequence number
SN	=	sequence number

I. Introduction

THE aerospace industry is gradually moving to integrated modular avionics (IMA) architectures, where existing bus structures such as MIL-STD-1553 are replaced by computer network protocols [1]. In this context, Avionics Full-Duplex Switched Ethernet (AFDX), standardized as Aeronautical Radio Incorporation (ARINC) 664 [2], is adopted by many avionics manufacturers and employed in aircraft such as Boeing 787 Dreamliner, Airbus A380/A350/A400M, ARJ2121, and Superjet [3]. AFDX is a switched network architecture based on IEEE 802.3 Ethernet and operates at 100 Mbps. Real-time communication is achieved with a connection-oriented structure together with traffic shaping both at the end systems (ESs) and switches. In addition, all switches and network interfaces have redundancy for strong fault tolerance support.

The amount of data carried in real-time mission-critical networks is increasing with the introduction of new devices, sensors, and audiovisual equipment such as cameras. To ensure the correct operation of control loops running on distributed devices, it is necessary not to exceed the end-to-end message delay limits, which are at the level of milliseconds. Furthermore, video and image transfers require high bandwidth. The fulfillment of these requirements must be checked at the design stage. Mathematical models [4] are used to provide worst-case bounds. Since such bounds are potentially not tight, feasible configurations might be rejected unnecessarily. Furthermore, the average performance of the network is also of interest. A low-cost solution for analyzing a given AFDX network and message set is using a network simulator. Such a simulator has to accurately model the standardized network components to achieve the correct results. In addition to analysis, it is possible to use the network simulator as a design tool by evaluating the outcomes of different parameter sets and iteratively determining the network configuration parameters before setting up the actual system. Furthermore, extremum values that can tightly represent the worst cases can be achieved by systematic exploration of a large input space.

An open-source platform facilitates the contribution of a community for extending and verifying the simulation models. A popular open-source network simulator constantly extended by a user community is OMNeT++ (extensible, modular, component-based C++ simulation library and framework) [5]. The network models that are created with this C++-based discrete event simulator consist of modules communicating with each other through message exchange [6]. In addition to the accuracy of the model, easy configuration and result collection of the simulation are desired features of the simulator framework. OMNeT++ provides its own integrated development environment (IDE) and file formats for configuring and running the simulation, which require the familiarity of the user.

This paper proposes the development and validation of a complete simulation framework for AFDX based on OMNeT++. The first contribution of the paper is a significant improvement of the ES and switch models in the previous OMNeT++ AFDX [7] in conjunction with a concise formal representation of the network parameters and timing bounds in the AFDX standard [2]. Specifically, we increase the fidelity to the AFDX standard, provide modules for detailed statistics collection, offer more configurable ESs and routing, replace obsolete libraries, and make the entire AFDX model compatible with OMNeT++ 6.0, which was released in April 2022. We compile our improved models into OMNeT++ 6.0. The second

Received 22 August 2023; revision received 14 December 2023; accepted for publication 7 February 2024; published online Open Access 22 March 2024.

*Senior Engineer, Department of Defense Technologies and Systems, Etimesgut; currently ICT Group, Applied Solutions, 5613 AM Eindhoven, Netherlands.

†Assistant Prof. Dr., Department of Electrical and Electronics Engineering, Etimesgut; murat.ucuncu@yahoo.com.tr; mucuncu@baskent.edu.tr (Corresponding Author).

‡Prof. Dr., Department of Electrical and Electronics Engineering, Çankaya.

contribution is analyzing the correctness of our improved model through structured analyses. To this end, we construct tailored experiment setups with computable results. The experiment parameters and results are presented in a consistent notation with our formal representation. Moreover, we repeat previously published AFDX simulation experiments conducted by using different simulators or methods. To the best of our knowledge, no validation of the OMNeT++ AFDX model was performed in the previous literature. Our model replaces the previous model and is published by the community [7,8]. The third contribution is a Python-based tool that we call ANCAT to augment our OMNeT++ models for a complete AFDX simulation framework. ANCAT enables the user to provide simulation configurations with standard *.xlsx files, run the simulation, and collect the results in an automatically generated report in a streamlined fashion without manual steps and specific file formats of OMNeT++. Configuration, running, and result collection for all experiments in this paper are carried out by ANCAT. To this end, our conclusive results show the correctness of ANCAT in addition to the OMNeT++ models. As the fourth contribution, we illustrate the practicability of our simulation framework with a detailed case study. We simulate and evaluate the performance of a realistic AFDX network that is based on the realistic Flight Management System [9,10] and extended with cameras and a data logger to cover contemporary Avionics applications. Hence, the message set includes periodic and sporadic sensor data and high-bandwidth video. We evaluate the components of the end-to-end latencies and the effects of sharing switch buffers by different flows. To this end, we present latency statistics that are collected per ES, per switch port, and per virtual link (VL). Our results show that real-time communication features of AFDX support the deterministic behavior of the network and small end-to-end frame delays as desired. Overall, we demonstrate the effective use of our simulation framework in investigating the relevant AFDX network parameters and performance metrics.

II. Background and Previous Work

IMA network architectures require high data rate (throughput), broadcast capability, bounded delay and jitter, redundancy management (RM), and implementation hardware [11]. IEEE 802.3 Ethernet is the layer 2 protocol that is frequently used in computer networks with standardized data rates of 10 Mbps/100 Mbps/1000 Mbps/10 Gbps. To this end, the use of Ethernet for real-time applications with different levels of compliance with commercial off-the-shelf (COTS) implementations is proposed [12,13].

Several Ethernet-based real-time network protocols exist that have potential use in avionics. Time-triggered protocol (TTP) is a shared-medium Ethernet protocol that achieves deterministic medium access with time division multiple access (TDMA) [11]. Such implementation requires precise time synchronization and static capacity allocation, which limit the maximum bit rate to 25 Mbps. TTEthernet [14] is a high-speed, connection-oriented, proprietary switched Ethernet network. The end-to-end delay bounds are achieved with very precise synchronization and complicated switch management. Time-sensitive networking (TSN) is an extension of TTEthernet with multiple IEEE standards to support Quality of Service (QoS). The configuration of TSN is more complicated compared to TTEthernet as it requires the timing of the switching through the use of gate control lists [15].

The AFDX communication protocol is based on 100 Mbps IEEE 802.3 Ethernet medium access control (MAC) and uses a compliant physical (PHY) layer [16]. AFDX comes forward with its full duplex, switched topology that is not affected by the inefficiencies of bus arbitration, 100 Mbps data rate, bounded latency, or ease of configuration compared with the legacy avionics bus protocols such as ARINC 429, ARINC 629, and Mil-Std-1553 [17]. Furthermore, AFDX provides compatibility with the standard network protocol stack supporting UDP/IP [1,11]. AFDX is standardized as ARINC Specification 664 Part 7 [2].

It is important to determine if a given AFDX network configuration, with its topology and traffic characteristics, meets the real-time requirements of the applications. To this end, the computation of end-to-end latency bounds of AFDX networks using network calculus is proposed [18,19]. Network calculus analysis is used as a step for optimization of the configuration parameters [20]. An important issue regarding such an analytical approach is the tightness of the bounds where pessimistic results can deem a feasible network configuration infeasible [21].

A. AFDX Protocol and Parameters

In this section, we present relevant AFDX specifications that are defined in [2]. The components of the AFDX network architecture are the switches (SWs) and the ESs. All links are point-to-point and full duplex. Each ES runs one or more partitions, which are the program units of an application. AFDX achieves the determinism required for the communication of real-time avionics systems by defining VLs. Each VL represents a single communication flow between a source partition and multiple receivers with a statically configured route. Furthermore, traffic shaping and service parameters are defined for each VL at the source ES and the switches on its path to achieve resource allocation and logical isolation. SW_i denotes the number of switches that VL_i frames go through. AFDX uses a modified Ethernet frame as shown in Table 1. The 16 bits of destination MAC address serve as the virtual link identification (VL ID). For a given AFDX payload P, the AFDX frame size is L, as determined by Eq. (1). The transmitted frame size S with the PHY overhead is given by Eq. (2):

$$L = \max(P, 17) + 47 \text{ B} \tag{1}$$

$$S = L + 20 \text{ B} \tag{2}$$

AFDX employs a dual-network architecture for reliability, with a redundant set of switches and links. Hereby, each ES sends two copies of each frame on channels A and B. Each AFDX frame carries a 1B sequence number (SN). If two frames with the same sequence number are received within SkewMax seconds at the receiver, the late duplicate is discarded. This is referred to as integrity checking (IC) and RM [22]. SN is incremented on each successive frame and wraps around in at least 255 ms for a bandwidth allocation gap (BAG) of 1 ms [2]. The ES stores the SN of the previously received frame of any VL_i in the variable previous sequence number (PSN_i). The received frame is accepted if its SN is in the interval $[PSN_i + 1, PSN_i + 2]$.

The application on the partition acts as a traffic source, TS_i that is mapped to VL_i. We indicate the minimum interval between two frame generation instants by TS_i with X_i. TS_i can be periodic or sporadic, where X_i is constant for periodic sources. The source ES applies traffic shaping with a regulator for VL_i that enforces BAG with the parameter BAG_i. Here, BAG_i is the minimum interval allowed between transmitting the first bits of two consecutive frames from VL_i out of the regulator. Frame j of VL_i $F_{i,j}$ can have a latency of $B_{i,j} > 0$ in the regulator to enforce BAG_i. BAG_i can take values of

Table 1 AFDX frame format

PHY Overhead		Ethernet Frame [64-1518]								PHY Overhead
Preamble	SFD	MAC Address		Type	Ethernet Payload				FCS	IFG
		Destination	Source		IP Structure	UDP Structure	*AFDX Structure*			
							Payload	*SN*		
7B	1B	*6B*	*6B*	*2B*	*20B*	*8B*	*17B-1471B*	*1B*	*4B*	12B

FCS, frame check sequence; IFG, interframe GAP; SFD, start of frame delimiter; SN, sequence number.

2^k ms, where $k = 0, \ldots, 7$. The maximum frame size values for VL_i are denoted by $L_i^{\max} \leq 1518$ B and $S_i^{\max} = L_i^{\max} + 20$ B. The ES selects the frame to transmit among the frames of different VLs using a scheduler. Here, we note that ARINC664 Standard [2] does not specify a policy for the ES scheduler for selecting the frame to transmit among the frames released by the regulators. The jitter $J_{i,j} \geq 0$ for a frame $F_{i,j}$ is defined as the additional time spent starting from the time instant that the regulator of VL_i released the first bit of $F_{i,j}$ until the time this first bit is transmitted by the ES [2]. An example arrival scenario in Fig. 1 shows the frames generated by traffic sources TS_1 and TS_2 on the same ES. Here, TS_2 generates $F_{2,2}$ after $F_{2,1}$ before BAG_2 expires. Accordingly, the regulator of VL_2 delays $F_{2,2}$ by an amount of $B_{2,2}.F_{1,2}$ experiences $J_{1,2}$ due to the transmission of $F_{2,1}$.

AFDX requires a bound for the jitter at the output of ES n, $J_n^{\max}$, to support deterministic, real-time network operation. This bound is called maximum admissible jitter and is defined for ES n with a line rate of C bps as in Eq. (3) [2]. Note that υ_n denotes the set of VLs that are scheduled by ES n.

$$J_n^{\max} = \max(J_{i,j}) \quad \forall\, F_{i,j}, i \in \upsilon_n \leq \min\left(40\ \mu\text{s} + \frac{\sum_{i \in \upsilon_n} S_i^{\max} \cdot 8}{C}\right) \tag{3}$$

Here, 40 μs is a typical value for the frame processing time. Note that this expression represents the worst case where all VLs have a frame ready at the ES scheduler with maximum frame sizes and the considered frame is scheduled as the last frame to transmit.

AFDX switches apply the token bucket shaper. In this algorithm, for each VL_i, the switch keeps an account denoted as AC_i each time a frame of S_i bytes that arrives at the switch at time t is accepted if $AC_i(t) > S_i$ followed by updating $AC_i(t) = AC_i(t) - S_i$. Otherwise, the frame is discarded, and $AC_i(t)$ is not changed. AC_i is updated as a function of time by $AC_i(t + \tau) = \max(\sigma_i, AC_i(t) + \rho.\tau)$ limiting the size of the burst that can be transmitted by VL_i to σ_i bytes. In Eq. (4), the calculations of σ_i and ρ_i are given, where $J_{i,\text{switch}}$ is expressed in seconds and represents the time window that a frame is assured to be located within.

$$\sigma_i = S_i^{\max} \cdot \left(1 + \frac{J_{i,\text{switch}}}{BAG_i}\right), \quad \rho_i = \frac{S_i^{\max}}{BAG_i} \tag{4}$$

We denote the end-to-end latency for a given frame $F_{i,j}$ starting from the time $T_{i,j}$ is in the regulator of the source ES until it is fully received at the destination ES. In a distributed real-time system, the messages should be delivered before the next message is generated such that the control loops' sampling times are respected. Consequently, $T_{i,j} < X_i$ is desired for all $F_{i,j}$ and is strictly required for system-critical VLs. We express $T_{i,j} = TC_{i,j} + TV_{i,j}$, where $TC_{i,j}$ is the constant component computed during the network and message set design. We denote the technological latencies for an ES and a switch by $T_{\text{es}}^{\text{tech}}$ and $T_{\text{switch}}^{\text{tech}}$, respectively. Then $TC_{i,j}$ consists of the technological delays of sending/receiving ES and SW_i switches together with the transmission delays on $SW_i + 1$ links of rate C as follows:

$$TC_{i,j} = 2 \cdot T_{\text{es}}^{\text{tech}} + SW_i \cdot T_{\text{switch}}^{\text{tech}} + \frac{(SW_i + 1) \cdot (S_{i,j}) \cdot 8}{C} \tag{5}$$

$TV_{i,j}$ consists of ES jitter $J_{i,j}$ and the delays on all the switch ports. It is variable and depends on the traffic characteristics of the VL_i and other VLs that are contending for network access on the same interfaces with VL_i.

B. Previous Work on AFDX Simulation Analysis

There are previous studies on the evaluation of AFDX using different network simulator tools. The NS2 simulator was used by Tian et al. [23] and Song et al. [24]. OPNET was used in a study run by Liu et al. [25]. They do not provide any detailed information regarding the message set or model parameters. IMA is investigated in a realistic AFDX case study run by Lauer [9] with a flight management system (FMS). This FMS is modeled and evaluated using OPNET by Safwat et al. [10] and end-to-end latencies are evaluated under different switch technological latency values. The work conducted by Annighoefer et al. [26] used real-time data as simulation input to verify selected aircraft system functions without constructing an expensive AFDX network in hardware. For that purpose, an AFDX model is established in MATLAB/Simulink with modified built-in blocks in a hardware-in-the-loop (HWIL) simulation. The simulation is verified and benchmarked with three different scenarios. Villegas Javier et al. [27] propose MATLAB/Simulink together with network calculus as an AFDX verification and validation framework. They first test the correctness of their models in simple use cases and then simulate a larger network with new kinds of components to show the viability of their approach. Han et al. [28] used a distributed integrated modular avionics (DIMA) model interconnected by an AFDX network in the UPPAAL model checker. In this work, they model the tasks running on the end-systems together with the network behavior. Their outcome is a feasibility analysis of a given task schedule.

In this paper, we focus on OMNeT++ [5], which is a powerful discrete event simulation environment for modeling communication networks of numerous different domains, and it is free to use for

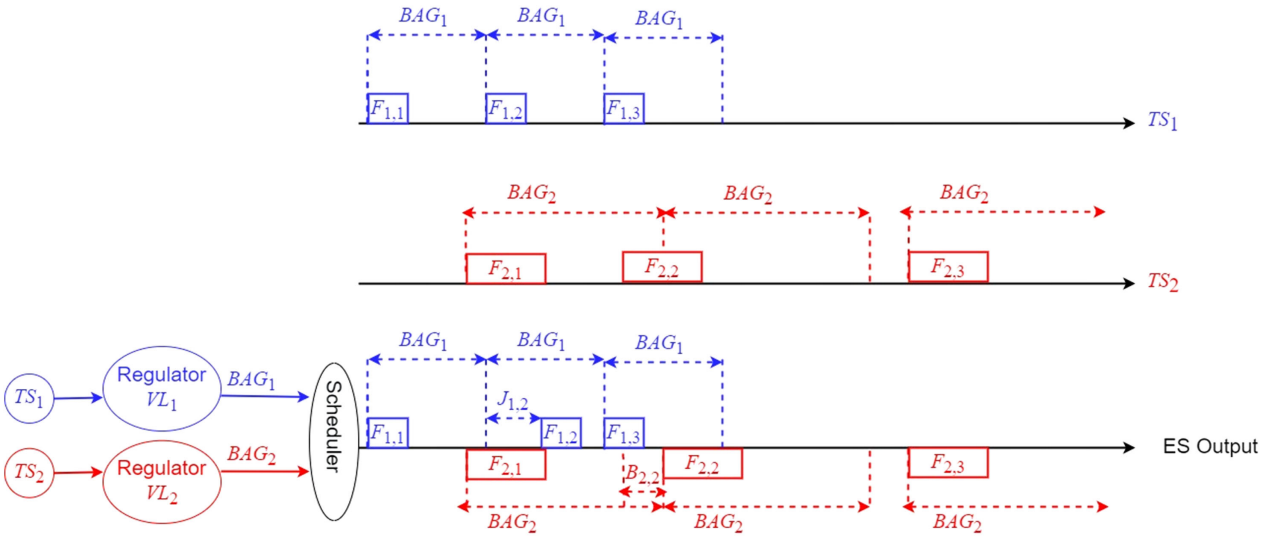


Fig. 1 AFDX ES traffic shaping.

nonprofit users. OMNeT++ is evaluated to outperform other widely used open-source network simulators such as NS2/NS3 in terms of runtime, memory requirements, and visualization ability [29,30]. For many specific areas, the OMNeT++ community [31] developed frameworks/packages to model popular protocols.

Steinbach et al., Yang et al., and Rejeb et al. [32–34] conducted AFDX simulation using OMNeT++. Steinbach et al. and Yang et al. [32,33] removed the timing and synchronization features of an existing OMNeT++ TTEthernet model to construct the AFDX model. They evaluate a small number of VLs and compare the mean and maximum latency values. Rejeb et al. [34] used the INET (an open-source OMNET++ model suite for wired, wireless, and mobile networks) Ethernet framework to build the AFDX model and compared the results with those obtained from TTEthernet hardware. Kultur and Bilge [35] provide a comparison of TTEthernet and AFDX using OMNeT++ (5.7 or older). They do not specify if they use the INET Ethernet framework of OMNeT++ or the previous AFDX module [7]. Their test cases are run on a simple network with a single switch and three ESs.

One can create simulation models in OMNeT++ by using C++ and Network Description (NED) languages. Class libraries in C++ define the functional behavior of the blocks. `*.ned` files define the network topology and default configuration values for the network components. Furthermore, OMNeT++ provides `*.ini` files that are not compiled to modify the configuration values for each simulation run. These file formats are specific to OMNeT++ and require users' familiarity.

The previous AFDX module that was published in the OMNeT++ community [7] was originally developed for OMNeT++ 4.0 and then ported to OMNeT++ 5.0. It uses the obsolete and third-party `queueinglib`, whereby the `AFDXMessage` class is derived from the `Job` class of `queueinglib`. The `trafficSource` block creates AFDX frames where the time between consecutive frame generations is specified either in `*.ned` or `*.ini` files. Furthermore, several features of the existing module have incomplete implementations and deviate from the AFDX specification [2]. In both ES and switch models, no per VL parameters are stored. Hence, `trafficSource` cannot specify per VL traffic parameters and the `redundancyController` has mismatches when checking the time difference between successive frames. The technological latency, $T_{\text{es}}^{\text{tech}}$ and $T_{\text{switch}}^{\text{tech}}$ are not modeled. `regulatorLogic` to enforce BAG shaping at the ES is introduced but not implemented. The outgoing port in the switch is determined by a function in the `VLRouter`. AC_i account manager is not implemented per VL. Note that `txQueue` and MAC blocks merely model the queuing delay without switch hardware latency and IFG in transmission. The invalid frame filtering is not implemented.

"Fault Tolerant Real-Time Ethernet Protocol" by Atik [36] is a preliminary work and proposes improvements for [7] by introducing the technological latency, BAG regulation at the ES, and enabling the configuration of different traffic sources per ES, implementing `VLRouter` as a table in the `*.ini` file. Certain per-VL features are also added. However, no verification of these updates was carried out and these updates were not submitted to the OMNeT++ community. Our work presented in Sec. III.A improves the model in the OMNeT++ community [7] and the preliminary work in [36]. To this end, we present Table 2 to summarize the issues [7,36] for better readability.

Table 2 Issues in [7,36]

Issue type	Issue
Missing AFDX feature	Technological latencies are not modeled [7]; they are modeled but not verified in [36].
	BAG regulation is not modeled in [7]; it is modeled but not verified in [36].
	Traffic policing in the AFDX switch is missing (token bucket algorithm is not implemented) [7,36].
	Most of the blocks in the simulation lack VL awareness [7,36].
	Integrity checker lacks PSN checking [7,36].
	Frame transmission times are not modeled [7,36].
Configuration limits	Changes in the topology can only be made by changing the code. For example: i) VL routing in the switch is hard coded. ii) The connections between ESs and switches are hard coded [7,36].
	It is not possible to have more than one traffic source in an ES; i.e., each ES can generate messages of one VL-ID [7,36].
	Measurement/logging-related facilities are very application-specific. They are not useful in different use cases [7,36].
Old software libraries	Some of the libraries that are used in the model are deprecated [7,36].
	Runs on the old OMNeT++ versions [7,36].
Availability	Not published to the community [36].

III. Proposed AFDX Simulation Framework

The main contribution of our paper is a complete AFDX simulation framework with OMNeT++. To this end, we first modify the OMNeT++ code to achieve more accurate results, as presented in Sec. III.A. Here, we note that these modifications include introducing new data types and classes that are recompiled into the latest OMNeT++ 6.0. The outcomes of these modifications are more realistic AFDX models and providing capabilities for more realistic network configurations and more detailed data collection.

Second, we propose the ANCAT tool in Sec. III.B. ANCAT enables the users to configure the simulation parameters with a simple .xlsx file instead of OMNeT++-specific file formats and get automatically generated readable reports of simulation results.

Finally, we design and present a series of custom-designed experiments to verify both the new OMNeT++ code and ANCAT. Such an approach is followed by Villegas et al. [27] for the verification of their Matlab/Simulink models. The parameters of these experiments are consistent with our formal description of AFDX in Sec. II. Here, we note that these experiments verify the resulting OMNeT++ models and the ANCAT tool as all experiments are configured using ANCAT. These experiments are presented in Sec. III.C.

A. New AFDX OMNeT++ Model

In this work, we improve AFDX Model for OMNeT++ [7] and the work in [36]. We implement our improved AFDX model in OMNeT++ 6.0, which was released on April 13, 2022 [37]. We note that we replaced `queueinglib` in [7,36], which is a very significant component for network simulation, with the new version that is included in OMNeT++ 6.0. Our model is approved to be published on the OMNeT++ community page [8]. We describe the new features of our AFDX model below. All these features are compiled into the OMNeT++ simulator. Note that the user does not need to recompile the simulator to run the experiments.

1. *Network Statistics Collection*

We designed a new class called `NetworkStatistics` with generic record-keeping. To this end, `createRecorder(recordType, key)` and `record(recordType, key, value2Record)` functions keep a unique record and update the existing record with a new value, respectively. These two functions use the types and functions of the built-in library vector and are called with an enum that defines the record type and a unique key. The record types can be extended. There are per-VL records and per-switch records such as `E2E_LATENCY_PER_VL` and `SWITCH_QUEUING_TIME_PER_VL_PER_SW`, respectively.

2. *Increasing the Fidelity to the AFDX Standard [2]*

We modify the derivation of the Job class such that the AFDX frame transmission on the Ethernet line consumes simulation time. To this end, different from the previous versions in [7,36], time will continue to pass when frames are serially transmitted on the lines that represent physical Ethernet cables, and the transmission time is included in the simulation results.

We modify the integrity checker for the precise check of SN to be in the interval $[PSN_i + 1, PSN_i + 2]$. The AC_i parameters ρ_i and σ_i are added as part of the configuration file. If the token bucket algorithm results in a credit shortage, the simulation logs this event and goes on

running. We introduce a new connector type Cable with explicit propagation delay and length parameters. We modify `trafficSource` by dividing it into new messageSource and `AFDXMarshall` constructs which are connected by Cable. Here, `messageSource` is a more generic source model that generates a `SubsystemMessage` with a specified length. `AFDXMarshall` converts SubsystemMessage into an AFDXMessage by using configuration parameters that are set by `*.ini/*.ned` files. This enables us to have a realistic representation of a partition with `messageSource` and `Cable`. An example use of this representation would be a sensor sending data from an RS485 interface at a rate of 115,200 bps.

3. Increasing the Configurability

The network topology cannot be modified by the `*.ini` file in [7,36]. We added a new type `ConnDef` to represent each connection in the topology. This type contains the features of a connection (ID of connection endpoints, type of connection, and cable length). Thus, an array of `ConnDef` types with a size of "number of switches+number of ES-1" defines the network topology. This new type can be added and configured through an *.ini file, which enables users to define the topology without changing the simulation code.

4. OMNeT++ Models

Our final OMNeT++ ES and switch models [8] are presented in Fig. 2. Here, `txQueueA` (and `txQueueB`) in the ES and `txQueue` in the switch maintain per VL FIFO queues as stated by [2]. The ES scheduler transmits the frames released by the `regulatorLogic` in the order that they are generated. If there is more than one frame of different VLs ready for transmission, the ES scheduler transmits them in the order of their VLIDs. We note that this is an arbitrary scheduling choice, as there is no related specification in [2].

B. AFDX Network Configuration and Analysis Tool: ANCAT

The OMNeT++ simulation workflow requires many disconnected and manual steps that require user intervention, such as modifying the `*.ini` file, running the simulation, retrieving the data, and visualizing it. In this work, we propose an AFDX Network Configuration and Analysis Tool (ANCAT). We publish ANCAT on GitHub [38]. ANCAT is a Python-based tool that takes the simulation configuration in a generic format such as *.xlsx, runs the simulation from the command shell, processes the simulation results, and presents them in a report [37]. ANCAT is composed of one batch file named `ANCAT_run.bat` and three Python scripts that are called `PreProcessor.py`, `SimProcessor.py`, and PostProcessor.py. The input `*.xlsx` file is composed of three main sections, such as Topology, Settings, and Message Set. The first section describes the connections between ESs that are to be constructed with `ConnDef` blocks. The second section contains network settings ($T_{\text{es}}^{\text{tech}}$, $T_{\text{switch}}^{\text{tech}}$, C, `SkewMax`, and the size of the per-VL queue in `RegulatorLogic`) and simulation settings (start and stop times). Finally, the third section includes details about the message set to be simulated and contains the following features for each message: VL-ID (VL_i), Partition ID, BAG_i, P_i, ρ_i, and σ_i.

`PreProcessor.py` computes the routing tables using the information in the `*.xlsx` input file and automatically creates the `*.ini` file. `SimProcessor.py` runs the simulation from the command shell without invoking the IDE and saves the collected record data that are formatted according to our `NetworkStatistics` class in `*.vec` and `*.vci` formats. Then `PostProcessor.py` creates a report in `*.pdf`, including plots. The report presents per VL and per switch statistics. `ANCAT_run.bat` runs `PreProcessor.py`, `SimProcessor.py`, and `PostProcessor.py` in sequence. For the rest of the paper, ANCAT outputs are used for the presented results, and the plots are automatically generated with ANCAT.

C. Simulation Models' Tests

We conducted several specifically designed experiments to verify the correctness of our AFDX model [8]. The line rate is $C = 100$ Mbps for all experiments. We select $T_{\text{es}}^{\text{tech}} = 32\ \mu$s and $T_{\text{switch}}^{\text{tech}} = 4\ \mu$s. Note that these are configuration parameters, and they are measured as they are configured in the simulation. These experiments are very controlled experiments, with periodic, computable results and repeating patterns of frames. The BAG values are limited to 1 ms. To this end, 1 s frame generation time is suitable for observing the network behavior. The network has two switches, where one switch is duplicated for redundancy.

We present the screenshots of OMNeT++ GUI for network modeling in Figs. 3–5 and Figs. 7, 8. In these figures, the redundancy of AFDX network is seen by duplicated switches for networks A and B. Each ES has 2 interfaces that connect to their respective switches in Network A and B. All ES and switch interfaces are full-duplex as in the AFDX standard. We assume that the switches are legacy crossbar or shared memory switches where $T_{\text{switch}}^{\text{tech}}$ represent the hardware delay excluding all buffering delays.

For each Experiment, we state the path of VL_i with an ordered tuple as follows: $(TX_i, sw_1 - p_1, sw_2 - p_2, \ldots, sw_M - p_M, RX_i)$. Here TX_i and TX_i respectively denote the transmitting and receiving ES for VL_i. The switch(es) and their ports that frames of VL_i are transmitted on the path are denoted by $sw_j - p_j$. We note that in all our experiments both networks have the same traffic. To this end, we state the paths and report the results from Network A only.

1. Experiment 1: ES Regulator Bag Enforcement

This experiment is designed to verify the `regulatorLogic` block. The ES latency is defined between the frame generation at `messageSource` and the output of `macA` in Fig. 2 (ES model). The experiment setup is shown in Fig. 3. There is a single VL, VL_1, with X_1 and BAG_1 to ensure that there is no jitter due to other VLs to observe the effects of BAG enforcement in an isolated way. The path for VL_1 is (ES0, switchA-Port 1, ES1). We investigated three scenarios. In all scenarios, we observe that all frames were sent out of the `regulatorLogic` complying with BAG_1. All frames of VL_1 are served in FIFO order.

In scenario 1, $X_1 = 1$ ms, resulting in 1000 packet generations, $BAG_1 = 0.5$ ms. Here, $X_1 > BAG_1$, where a constant arrival rate lower than the service rate results in no queuing. All frames are sent out of the `regulatorLogic` periodically with $X_1 = 1$ ms and the measured ES latency is the same as the technological latency ($T_{\text{es}}^{\text{tech}}$).

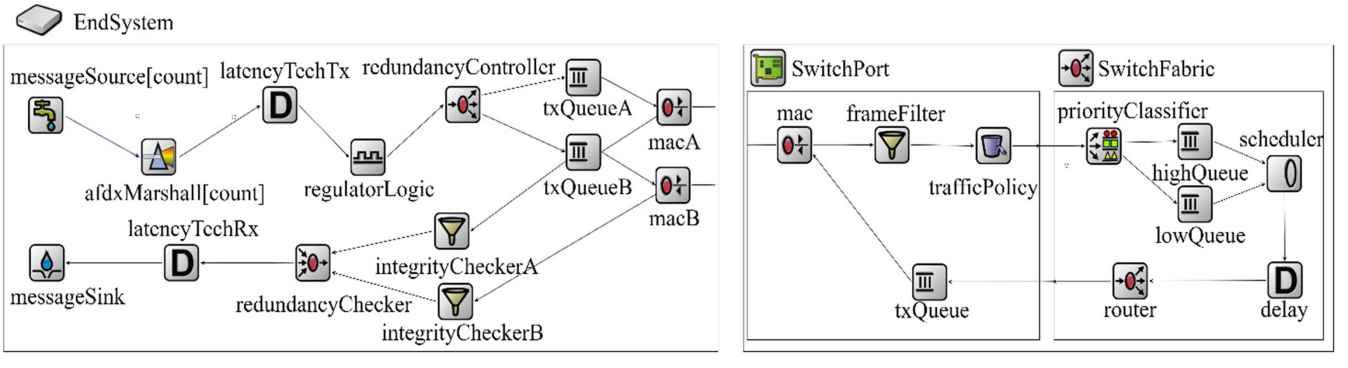


Fig. 2 OMNeT++ models.

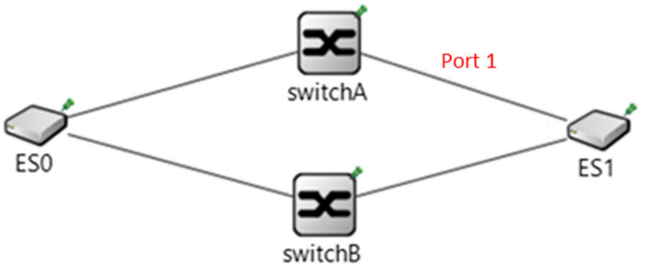


Fig. 3 Setup for Experiment 1.

Table 3 Scenario characteristics

Scenario	No. of VLs	Interarrival time, X_1, ms	BAG, ms
1	1	1	0.5
2		0.5	1
3		1 ± rand (0, 0.2)	1

In scenario 2, $X_1 = 0.5$ ms, resulting in 2000 packet generations, $BAG_1 = 1$ ms. In this scenario, $X_1 < BAG_1$, where a constant arrival rate greater than the service rate results in an unbounded increase in the queue size. All frames are sent out of the `regulatorLogic` periodically with $BAG_1 = 1$ ms. The ES latency of the first frame is equal to $T_{\text{es}}^{\text{tech}}$ and then, the latency for each frame increases linearly until the last frame with 1 s latency. The last frame is generated at $t = 1$ s and it takes a total of 2 s to transmit 2000 frames.

In scenario 3, X_1 is randomly selected with $X_1 = 1 \pm \delta$ ms, $\delta \leq 0.2$ ms. We show the histogram of the frame interarrival times in Fig. 4a to show the randomness of X_1. The resulting latency with $BAG_1 = 1$ ms is shown in Fig. 4b. In this scenario, the regulator latency fluctuates around 3 ms after the queue reaches a stable size. The scenario characteristics can be seen in Table 3.

2. *Experiment 2: ES Jitter*

In this experiment, there are four identical VLs with $X_i \leq BAG_i = 1$ ms, $S_i = 1250$ B, resulting in (1250.8 bits/100 Mbps) = 100 μs frame transmission time. The experiment setup is presented in Fig. 5. The path for all VLs are (ES0, switchA-Port 1, ES1). All VLs produce periodic frames simultaneously. All $F_{i,j}$ are simultaneously released by the regulators of VL_i without any delay. As the generation time is the same for all j, the ES scheduler transmits the frames in the order of their VL IDs according to our implementation described above. The ES latency of each VL_i is constant and equal to $T_{\text{es}}^{\text{tech}} + (i-1) \cdot 100$ μs, which is the sum of the technological latency and the transmission times of frames that are scheduled before VL_i frames complying with $J_n^{\max}$ in Eq. (1).

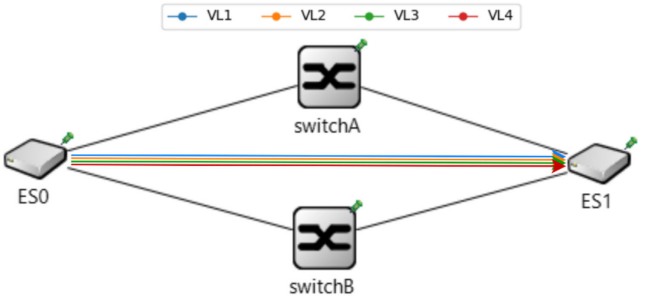


Fig. 5 Setup for Experiment 2.

3. *Experiment 3: Switch Account Management*

This experiment is designed to verify the `trafficPolicy` block. The experiment setup is given in Fig. 6.

ES 0 is the source of 7 VLs. For all VL_i, $BAG_i = 1$ ms and $S_i = 10$ kbits, resulting in the frame transmission time of 100 μs and $\rho_i = 10$ kbits/1 ms = 10 Mbps as defined by Eq. (2). VL_1 is destined to

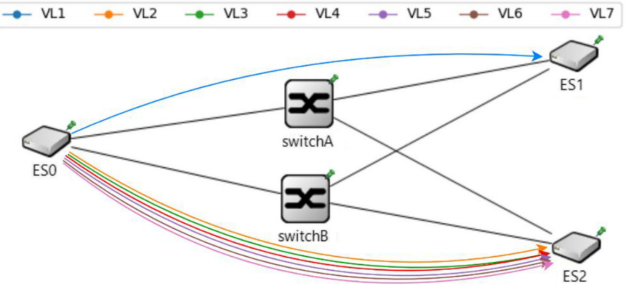
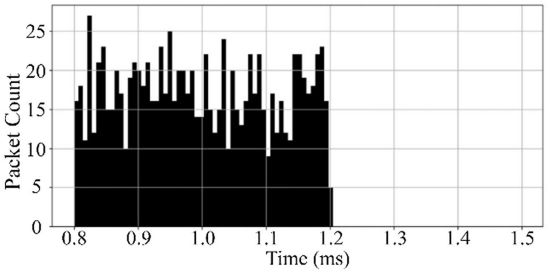
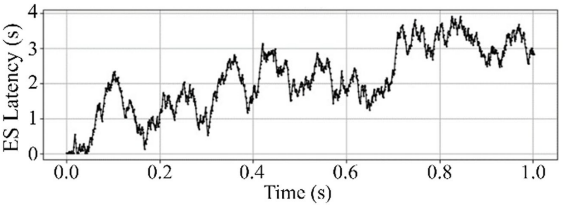


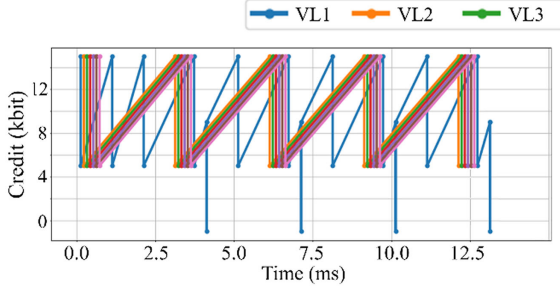
Fig. 6 Setup for Experiment 3.



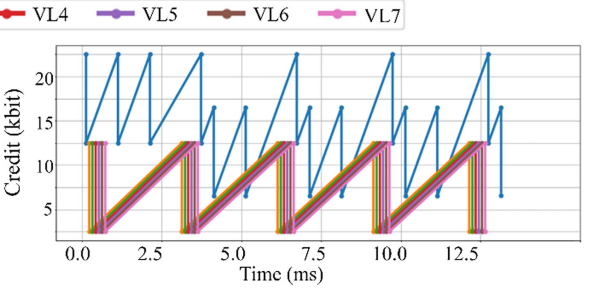
a) Experiment 1: Random frame arrivals at ES



b) Experiment 1: ES latency for random frame arrivals



c) Experiment 3: Switch account management with frame drops



d) Experiment 3: Switch account management without frame drops

Fig. 4 OMNeT++ model verification experiment.

ES 1. The path for VL_1 is (ES0, switchA-Port 1, ES1). $VL_2 - VL_7$ are destined to ES 2 on a different switch port. The path for $VL_2 - VL_7$ is (ES0, switchA-Port 2, ES1). X_i values are fixed with $X_1 = 1$ ms, $X_2 = X_3 = \ldots = X_7 = 3$ ms. $F_{1,1}$ and $F_{1,2}$ arrive at the switch with an interval of 1 ms and get served. $F_{1,2}$ arrives at the switch delayed by the 600 μs jitter from ES 0 due to $F_{i,1}$, $i = 2, 3, \ldots, 7$. When $F_{1,3}$ is served, AC_i is decremented by 10,000 followed by the arrival of $F_{1,4}$ after 300 μs. Here, note that AC_i is the accumulated credit of a queue.

We show the change of AC_i values for all VLs in Figs. 4c and 4d for the first 15 ms of the experiments, as they have a repeating pattern due to the periodic traffic. In Fig. 4c, $\sigma_i = 15{,}000$ and $AC_i(0) = 5000$ for all VLs. When $F_{1,3}$ is served, AC_1 decreases to 5000. When $F_{1,4}$ arrives, the credit account AC_1 is 8000, which is less than $S_{1,4}$ ($AC_1 = 5000 + \rho_1 \cdot 300\ \mu s = 8000$). Hence, $F_{1,4}$ is dropped. We indicate the drop of the frame with $AC_1 = -2000$ in Fig. 4c only for demonstration purposes. Here, we note that, in the case of frame drop, AC_1 is not updated and it is not negative in the actual AFDX implementation. This arrival-and-drop pattern repeats periodically. The same arrival scenario is repeated with $\sigma_1 = 20{,}000$ and $AC_1(0) = 10{,}000$ in Fig. 4d without any packet drops with the expected numerical results. This time AC_1 decreases to 10,000 after $F_{1,3}$ is served and increases to 13,000 when $F_{1,4}$ arrives. Serving $F_{1,4}$ decreases AC_1 to 3000.

4. *Experiment 4: Switch Latency and Buffer Management*

In this experiment, we demonstrate the correctness of the queuing behavior of the model by using Little's Law [39]. The experiment setup is given in Fig. 7 below.

There are eight VLs, where for all VL_i, $BAG_i = 1$ ms and $S_i = 10$ kbits, resulting in the frame transmission time of 100 μs. The frame generation times have randomness with $X_i = 1 \pm \delta$ ms, $\delta \leq 0.2$ ms. VL_1, VL_2, VL_3, and VL_4 are from ES 0 to ES 2. VL_5, VL_6, VL_7, and VL_8 are from ES 1 to ES 2. According to Little's Law, $N = \lambda \cdot T$. Here, N is the mean number of frames in a queue, λ is the average arrival rate in frames/s and T is the average time spent by a frame in the system in seconds. The simple form of Little's law assumes no frame drops; hence, the token bucket parameter is set to a large value of $\sigma_i = 5.S_i$. We measure N by counting the entering and exiting frames to txQueue in the switch port model in Fig. 2. We measure T for each frame as the difference between entry and exit times to txQueue block excluding the frame transmission time of 100 μs. Finally, we measure the average frame rate λ for ES 2 by counting the number of frames arriving at the messageSink block in the ES model in Fig. 2 divided by the simulation time. We collected the following results: 8019 frames are collected at the messageSink during a total simulation time of 1.0092 s, yielding a calculated $\lambda =$ 7945 frames/s. The measured T is 38.202 μs. The average frame queuing delay T is measured to be 38.202 μs. The measured N is 0.3035 frames, which is consistent with 7945 frames/s $\times$ 38.202 μs.

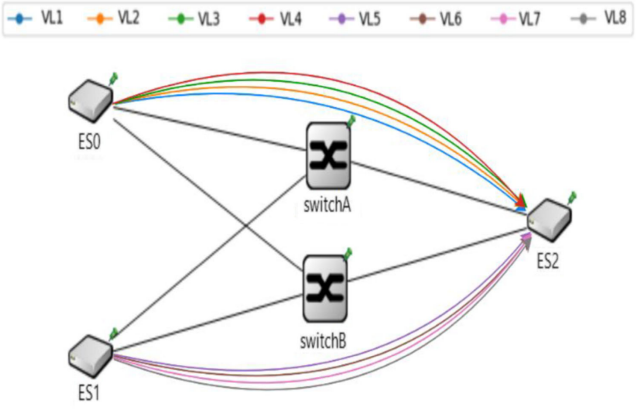


Fig. 7 Setup for Experiment 4.

5. *Experiment 5: SkewMax Test for Redundancy Management*

For this experiment, we used the same setup as the one in Experiment 4. Additionally, we introduce a test block that we call skewMax-Tester between the redundancyChecker and the integrityCheckerA/integrityCheckerB modules of the receiving side of the ES model in Fig. 2. The purpose behind this extra block is to be able to assess the simulation's behavior when a redundant frame is delayed for more than skewMax seconds. When identifying redundant frames, the time difference between frames from switch A and switch B is measured. If the skew between the frames is less than the skewMax value, the later frame is identified as a redundant frame and discarded, but otherwise, it will be accepted as a new frame [2].

In this experiment, `SkewMax = 10 ms`. We delete a number of frames of VL_1 to increase the time difference between successive frames. After a time period that is more than 10 ms, it retransmits the latest frame. We observe that `redundancyChecker` accepts this copy frame one more time as it is received after SkewMax.

6. *Experiment 6: Comparison with OPNET*

Our final verification experiment repeats the evaluation of a realistic avionics network that was simulated using the OPNET simulator [40]. IMA is investigated over a realistic AFDX case study in [9] with a flight management system (FMS). The network consists of 9 ES and 5 switches that are connected in a star topology. This FMS is modeled and evaluated using OPNET [10].

We simulate the FMS using our OMNeT++ AFDX model and use ANCAT for the configuration and result collection. There are 12 VLs with BAGs that vary from 8 to 64 ms. Safwat et al. [10] presented the end-to-end latency for three selected VLs under switch technological latency values of 140 and 16 μs. When the technological latency is 140 μs, the mean end-to-end latency values in [10] and the values that we measure using our models are 477, 154, 492, and 442, 151, 454 μs, respectively. When technological latency is 16 μs, the latency values of [10] and our model are 194, 33, 92, and 194, 27, 82 μs, respectively. Hence, we achieve very close values to those measured in the OPNET simulator, as summarized in Table 4.

IV. Evaluation of a Realistic AFDX Network with OMNeT++ AFDX Model

In the previous section, we demonstrated the correctness of our simulation framework with custom-designed experiments that produce computable results, together with repeating the experiments and achieving very close results to those obtained with another simulator in the literature.

In this section, we demonstrate the viability of our complete simulation framework by investigating the real-time performance of AFDX for an avionics network with a realistic and representative message set and topology. Here, we note that avionics networks are strictly proprietary and dependent on the physical properties of the aircraft. To the best of our knowledge, the FMS introduced in [9,10] is the only publicly available example of an avionics network. We extend this network with cameras to address the contemporary requirements of video transmission. Furthermore, we added a data logger, which is a frequently used avionics component. We emphasize that all these added components significantly increase the traffic on the network to test AFDX performance under demanding conditions. This realistic network has VLs with periodic and sporadic traffic generation and fixed or variable frame

Table 4 Comparison of VL-ID data against a different simulator (OPNET)

	End-to-end latency, μs			
	SW type 1		SW type 2	
VL-ID	Safwat et al. [10]	Proposed model	Safwat et al. [10]	Proposed model
0 × 7	477	442	194	194
0 × 9	154	151	33	27
0 × B	492	454	92	82

sizes. Furthermore, we evaluate a variation of this network by increasing the frame generation and BAG values of a subset of VLs.

Switch ports transmit different mixtures of these traffic types. AFDX is a switched Ethernet network where queuing at the switch ports can affect the end-to-end latency.

We note that an extensive and generalized AFDX performance analysis is not in the scope of this work. This case study demonstrates that our simulation framework can generate per VL traffic with desired and realistic characteristics. We present the measurements of per ES, per switch port and per VL latency values. The measurement and statistics collection capabilities of our simulation framework enable the user to locate the significant latency components. Furthermore, it is possible to observe and evaluate the effects of VLs that are sharing a switch output port.

A. Network Topology and Message Set Properties

The realistic network that we evaluate is presented in Fig. 8 with the numbering of the switch ports. Different from [9,10], we model all ES with technological latency. Here, the technological latency values are selected as 40 μs [2] and 140 μs [10] for the ES and the switches, respectively. σ_i and ρ_i values in the switches are selected accordingly, as defined in ARINC 664 P7-1 [2] with $J_{i,\text{switch}} = 500$ μs. There are 14 ES, 5 duplicated switches, and 30 VLs.

Here, ES0 and ES1 are GUI modules containing a keyboard unit (KU) and multi-function display (MFD) subsystems, respectively. ES2 and ES3 are flight manager (FM) modules. ES4 and ES5 modules contain air data inertial reference units (ADIRU), which receive data from ES6 and ES7, namely, remote data concentrator (RDC), which collects sensor data. In this network, some ES are sending periodic data to each other, and some ESs are working in a command-response fashion. Accordingly, RDCs send the data that is gathered from sensors to the ADIRUs, and ADIRUs direct those data to the FM after making additional calculations. In one possible application, when a user requests the estimated arrival time and distance to the target from the user interface, the responsible ES sends a request to the FM, which results in another request from the FM to the navigation database. Finally, the FM calculates the requested information by combining the navigational data with sensor data that are already gathered periodically and sends it to the ES that is responsible for the user interface, again periodically. There are two cameras, one data logger, and two actuators. Each video stream is split into three VLs in order not to exceed the dedicated bandwidth of a VL. We assume that the cameras generate H.264 video data in packetized elementary stream (PES) [41] with 188 B PES data units. A maximum length Ethernet frame can fit seven PESs, yielding a video payload of 1316 B. We assume a 20 Mbps video stream is transmitted in three VLs. This yields a minimum inter-frame time of 1.632 ms. We add ES12 and ES13 for demonstration of receiving maximum-sized Ethernet frames from the FM.

Lastly, the data logger ES listens to all messages from all VLs except from the cameras. Note that if a VL has two destinations, only one frame is transmitted out of the source ES, and duplication is performed by SW_0. For each VL that has ES as a second destination, we separately collect the per-VL statistics for frames destined for ES9 and the other destination ES. This enables us to see the effects of different numbers of switches on the paths to two different ES for the same VL. Furthermore, Port 9 of SW_0, which is connected to ES9, carries approximately 50% of the network traffic, which increases the queuing delay at this port. We note that data sent to the logger is not system-critical. For the rest of the paper, we denote the frames of VL_i that are not sent to the logger as system-critical traffic.

The message set properties and the VL assignment are shown in Table A1. For each VL_i, the type of the traffic source TS_i is indicated with P for periodic and S for sporadic. For sporadic sources, the time between two consecutive frames changes uniformly and randomly between $X_i^{\min}$ and $X_i^{\max}$ indicated with parenthesis. Here, we note that $TC_{i,j}$ in Eq. (5) is a fixed value represented by TC_i for VLs with fixed P_i values. The payload sizes of the frames of VLID 0×18 and 0×19 are uniform and random between the indicated $P_i^{\min}$ and $P_i^{\max}$ values listed in the parenthesis in Table A1. TX_i and RX_i are transmitting and receiving ES nodes respectively. BAG_i and X_i are in ms. P_i is in bytes.

We define the maximum load for VL_i to be $G_i^{\max} = S_i^{\max} \cdot 8/(X_i^{\min} \cdot 1000)$ Mbps. We denote the set of VLs that are transmitted on port p by υ_p with the cardinality $|\upsilon_p|$. We define the total maximum load on port b with $G_p = \sum_{i \in \upsilon_p} G_i^{\max}$. We present the calculated values of G_p and $|\upsilon_p|$ for our network in Table A2. Note that frames are transmitted in both directions for $p = 14$ and $p = 16$. The switch ports have shared buffers for all VLs.

B. Experiments and Results

We simulate the realistic network with our AFDX OMNeT++ model [8] that we present in Sec. III. All simulation configurations and result collections were carried out by ANCAT. The simulation runs for 50 s of simulation time. The measured mean values for all quantities are within the $\pm 3\%$ confidence interval of the true mean.

1. *ES Scheduling Jitter*

Table 5 shows the maximum measured ES scheduling jitter for each ES_n indicated by $J_{n,\text{sim}}^{\max}$ together with the maximum jitter bound $J_n^{\max}$ calculated as in Eq. (1). We first observe that $\forall\ n$, $J_{n,\text{sim}}^{\max} < J_n^{\max}$. We note that $J_{n,\text{sim}}^{\max}$ is proportional to $J_n^{\max}$ showing that the number of VLs and their frame sizes at each ES should be considered at the design stage when planning the end-to-end delay budget for a given VL_i.

2. *Mean and Maximum Queuing Delays of Switch Ports*

We measure the mean and maximum queuing delays on all switch ports. The mean queuing delays on all ports except for Port (9,0) are less than 6 μs. The mean queuing delay measured at Port (9,0) is 104 μs due to the very high loading in the network (Gp, $|Vp|$) and VLs with

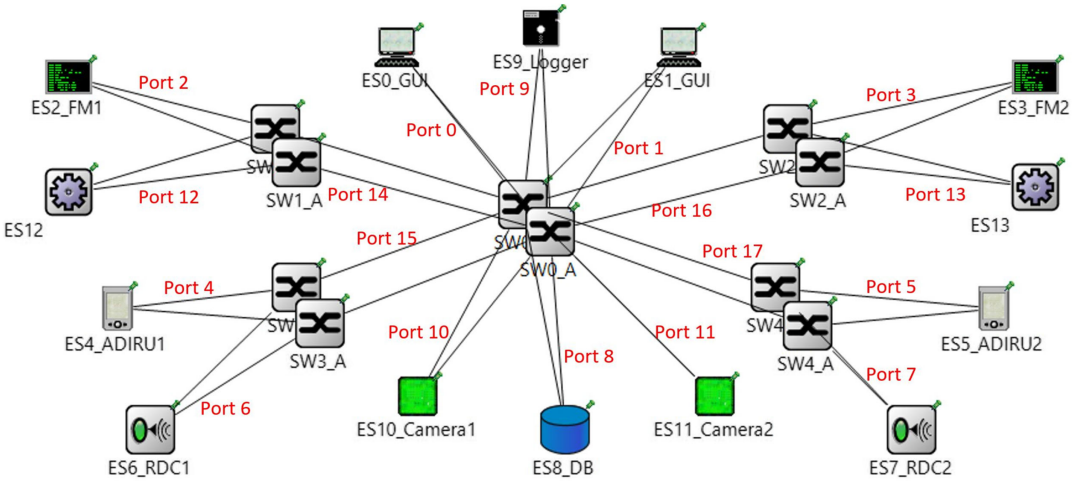


Fig. 8 Realistic avionics network topology.

Table 5 Measured ES jitter ($J^{\max}_{n,\text{sim}}$) and jitter bounds ($J^{\max}_{n}$) in μs

ES	$J^{\max}_{n,\text{sim}}$	$J^{\max}_{n}$
0	47	55
1	47	55
2	296	398
3	283	398
4	49	59
5	45	59
6	283	293
7	262	293
8	81	123
10	259	372
11	257	372

variable size frames. Furthermore, the queuing delay shows a high variance at Port (9,0). When we evaluate the maximum observed queuing delays, we see that Port (9,0) has the highest value with 848 μs. The next highest maximum queue delays are observed on ports with the next largest values of $(Gp, |Vp|)$. Here, we note that three high-$G_i^{\max}$ VLs on Port (0,0) are carrying data from a single video stream data. To this end, their frames are sequentially produced and transmitted without any significant queuing delay. Port (1,0) has the same configuration. We measure 350 μs, 349 μs for ports (2,1), (3,2) and 238 μs, 240 μs for ports (14,0), (16,0) respectively. We emphasize that these maximum values are very rare. These results show that the traffic shaping of AFDX at the ES and switch ports mitigate the randomness of the queuing delays in conventional Ethernet switched networks and increases the determinism of the real-time communication.

3. *Logger Traffic*

We observe that $T^{\text{mean}}_{i,\text{sim}}$ for frames destined for the logger is usually smaller as compared to the system-critical VL_i frames since there are fewer switches on the path. However, $T^{\max}_{i,\text{sim}}$ values of the frames destined to the logger show the effects of the queuing at Port (9,0) and they are mostly larger with respect to the values we observe in Fig. 9 for the same VL.

4. *Effect of Shared Switch Ports*

We select four VLs for a detailed evaluation of $T_{i,\text{sim}}$. VL_{0xB} to ES12 and $VL_{0\times16}$ to ES3 have periodic traffic sources with fixed-size frames. VL_{0xB} is transmitted on switch Port (16, 0) with another periodic VL_{0xE}. There was an insignificant change in the end-to-end latency measurements for VL_{0xB}, 24,998 samples have the same 464 μs latency. Two samples have 466 μs and 473 μs latency. We present the box plots of other VLs in Fig. 10. We divide the range between the measured $T^{\max}_{i,\text{sim}} - T^{\min}_{i,\text{sim}}$ into eight intervals and present the number of $T_{i,\text{sim}}$ values in each interval. $VL_{0\times16}$ is periodic with fixed-size frames. However, $VL_{0\times16}$ frames are transmitted on switch Ports (17, 4) and (16, 0) are queued with frames of other VLs with sporadic traffic and variable sizes resulting in end-to-end delay variance. Almost all the frames observe a delay of 574 μs. However, due to the nondeterministic queuing, there are few outliers that reach a maximum delay of 890 μs. $VL_{0\times10}$ has a sporadic traffic source with fixed-size frames and $VL_{0\times19}$ to ES1 has a periodic traffic source with variable-size frames. We observe that $T_{i,\text{sim}}$ for $VL_{0\times19}$ is larger distributing close to 100% of the measurements between 722 and 850 μs. The median is at 764 μs. There are a few outlier measurements with values up to 907 μs. We further observe that both $T^{\max}_{i,\text{sim}}$ values and the variance of the latency for the frames of $VL_{0\times10}$ to ES9 are larger than the frames of $VL_{0\times10}$ to ES2. This is due to the large number of VLs queued at Port (9, 0). The mean delay is around 791 μs. Almost close to 100% of the measurements are below 1000 μs. There are very few outlier frames with latencies up to 1500 μs. Almost all frames of $VL_{0\times10}$ to ES2 have a latency of 997 μs, which is very close to the mean latency. The maximum latency is 1200 μs. We observe that the latency of the VLs with fixed payload sizes and shared interfaces with other fixed payload VLs are with very small variance and the maximum latency is observed very rarely. We conclude that sporadic frame generation, variable frame sizes, and sharing interfaces with such irregular traffic increase the delay variance.

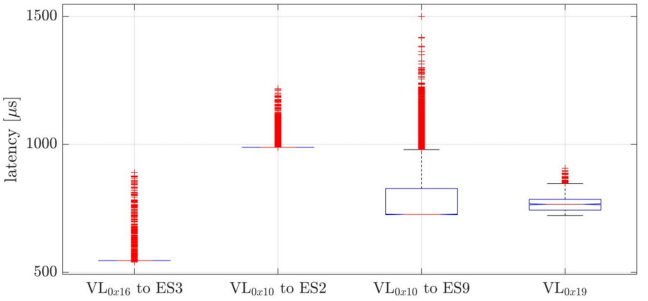


Fig. 10 Box plots for $T_{i,\text{sim}}$ measurements. The latency ranges are in μsv.

5. *Increased VL Traffic*

In the last part of our evaluation, we decrease the X_i values of VLs $0\times10, 0\times11, 0\times12$, and 0×13 from the (2,5) ms range to the (1,5) ms range. BAG_i are also decreased from 2 to 1 ms. We would like to note that this modification targets a higher worst-case load with respect to the original FMS network without significantly changing the mean values. The number of frames generated by the modified VLs for the same simulation duration increased by around 17% for these VLs.

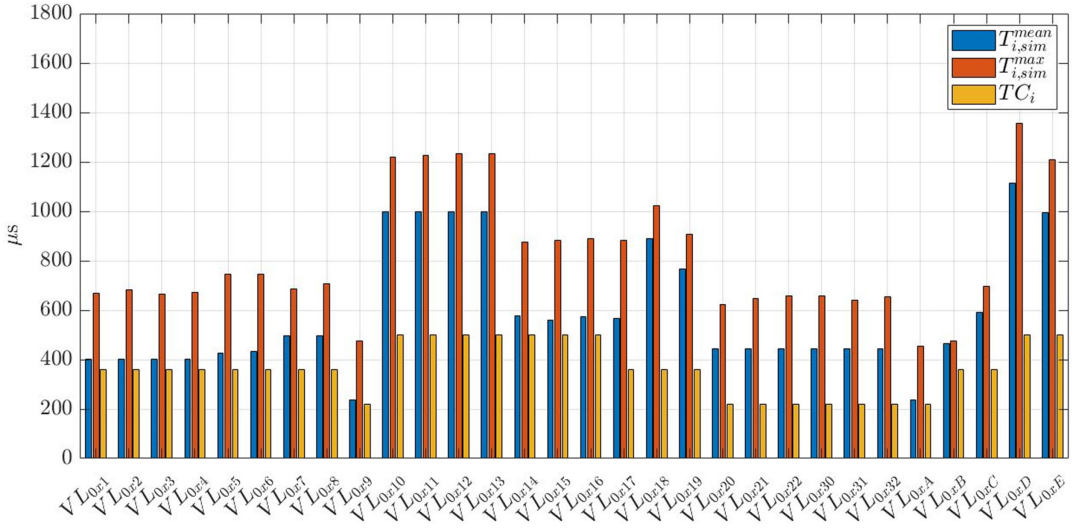


Fig. 9 Measured $T^{\text{mean}}_{i,\text{sim}}$, $T^{\max}_{i,\text{sim}}$, and TC_i for time critical frames.

Furthermore, this modification does not alter the maximum jitter bound $J_n^{\max}$ calculated as in Eq. (1) for the original FMS network, as the payload and the resulting frame sizes are not modified. The transmitting ES for VLs 0×10 and 0×11 is ES 6 and for VLs 0×12, 0×13 is ES 7. The measured maximum Jitter values, $J_{n,\text{sim}}^{\max}$ for these ES are $J_{6,\text{sim}}^{\max} = 283$ ms (5% increase compared to the original FMS) and $J_{7,\text{sim}}^{\max} = 262$ ms (the same as the original FMS), respectively. We would like to note that the remaining $J_{n,\text{sim}}^{\max}$ values are almost the same as the original FMS, with less than 1% difference due to the sporadic traffic generations, which are smoothed out by the number of samples.

This increases the G_p values of the relevant switch ports as follows: (2,1), $G_p = 24.5$ Mbps; (3,2), $G_p = 24.5$ Mbps; (9,0), $G_p = 74.3$ Mbps; (14,0), $G_p = 30.6$ Mbps; (15,3), $G_p = 24.5$ Mbps; (16,0), $G_p = 30.6$ Mbps; (17,4), $G_p = 24.5$ Mbps. No frames were dropped. One interesting observation is that although the computed maximum load that represents the worst-case on port (9,0) increases, the actual measured mean and maximum queuing delays almost stay the same at values of 104 μs and 848 μs. Note that these latencies are still less than the decreased X_i and BAG_i of 1 ms. To this end, no new frames of VL_i arrive before the service of the frames from the queue is completed, which results in similar maximum queuing latencies when X_i and BAG_i are 2 ms. We note that the mean queuing delays at all ports except for (9,0) stayed at their previous low values. The maximum queuing latencies of ports (2,1), (3,2) are increased to 386 μs, 409 μs and ports (14,0), (16,0) to 295 μs, 273 μs, respectively.

$T_{i,\text{sim}}^{\text{mean}}$ values differ within ± 10 μs compared to the original FMS network for all VLs and all destinations, including the logger. The observation that some $T_{i,\text{sim}}^{\text{mean}}$ values decreased shows the effect of the sporadic traffic sources and variable frame size VLs on the results. The difference in $T_{i,\text{sim}}^{\max}$ values has a more visible trend. $T_{13,\text{sim}}^{\max}$ to ES3 and $T_{11,\text{sim}}^{\max}$ to ES9 increase by 80 μs and 165 μs, respectively. It is interesting to see that $T_{8,\text{sim}}^{\max}$ to ES3 and $T_{8,\text{sim}}^{\max}$ to ES9 also increased by 107 μs and 161 μs due to the increased queuing delay of the shared switch ports.

This experiment demonstrates the significance of simulating the network before actually deploying it. On the one hand, the increased worst-case switch port loads have almost no effect on the measured mean latencies. On the other hand, the maximum link utilization (excluding the logger) is around 31%, which is not a very high value for a non-real-time computer network. However, the simulation results show that for this message set configuration, $T_{i,j} < X_i$ is not satisfied for VLs 0×10, 0×11, 0×12, 0×13 possibly violating the safety measures.

V. Conclusions

AFDX networks are used for safety-critical applications with bounded latency and guaranteed bandwidth requirements. To this end, network configuration parameters such as VL assignments, network topology, traffic shaping parameters at the ESs, and switches should be defined by considering performance metrics including line utilization, average and worst-case timings, switch queuing latencies, and buffer occupancies. On the one hand, there are analytical studies that feature network calculus to compute end-to-end latency bounds for AFDX networks. However, if it is possible that the computed bounds are too pessimistic, resulting in dismissing an actually feasible network configuration, simulation is a low-cost solution for analyzing a given AFDX network topology and message set at the design stage. A correct and accurate model can provide useful average values for the measured metrics. The network designer can iteratively determine the network configuration parameters by evaluating the outcomes of different parameter sets before setting up the actual system. Furthermore, systematic exploration of a large input space can achieve extremum values that can tightly represent the worst cases. In addition to the accuracy of the model, easy configuration and result collection in the simulation are desired features of the simulator framework, rather than complicated IDEs or simulator-specific scripting and file formats.

Acknowledging the benefits of an accurate and easy-to-use simulator, the main contribution of this paper is the development and analysis of a complete simulation framework for AFDX based on OMNeT++. The proposed AFDX model is implemented with the latest version of OMNeT++ and is published on the OMNeT++ community page. The model increases the fidelity of the previous AFDX OMNeT++ model by including specific features of VLs, providing account management in the switches, adding a new link and traffic source structure that allows us to accurately model different traffic sources, and implementing a module for detailed data collection. We present a complete simulation framework with a Python-based tool that enables simulation configuration through the widely used *.xlsx file format, runs the simulation, and automatically generates readable reports of results. The framework as a whole is analyzed with custom-designed experiments with computable results and repeating the experiments of a previously published work that uses another simulator.

Following the experiments, we conduct a detailed analysis of a realistic AFDX network topology and message set to demonstrate the effective use of our simulation framework. We measure the jitter in each ES and compare it with the maximum specified by the standard. We measure the fixed and variable components of the end-to-end delays for each VL and queuing delays at switch ports.

Our results for this particular case study show that AFDX can achieve end-to-end delays with small variances thanks to the traffic shaping at the ESs and switches without employing the advanced synchronization and timing controls of TTEthernet. The results also show that network designers should consider the frame size variance, sporadic sources, and the number of VLs that share the ES and switch port interfaces to ensure deterministic end-to-end network delays.

We are currently developing an analytically tight worst-case response time calculation for AFDX networks. The developed method will be integrated into our simulation framework. Our model currently uses FIFO queues in switches. Different scheduling algorithms can be added to the framework in the future. As a final enhancement, instead of an Excel sheet, a more graphical user interface can be provided.

Appendix: Realistic Avionics Network Test Data

Table A1 Message set for the realistic avionics network

i	TS_i	TX_i	RX_i	BAG_i, ms	X_i, ms	P_i, bytes
0×1	S	0	2,9	32	(50,100)	28
0×2	S	0	3,9	32	(50,100)	28
0×3	S	1	3,9	32	(50,100)	28
0×4	S	1	2,9	32	(50,100)	28
0×5	S	2	8,9	16	(60,100)	78
0×6	S	3	8,9	16	(60,100)	78
0×7	S	8	2,9	64	(100,150)	453
0×8	S	8	3,9	64	(100,150)	453
0×9	P	6	4,9	4	4	17
0×10	S	6	2,9	2	(2,5)	1471
0×11	S	6	3,9	2	(2,5)	1471
0×12	S	7	2,9	2	(2,5)	1471
0×13	S	7	3,9	2	(2,5)	1471
0×14	S	4	2,9	32	40	53
0×15	S	4	3,9	32	40	53
0×16	P	5	3,9	32	40	53
0×17	P	5	2,9	32	40	53
0×18	P	2	0,9	8	40	(683,1183)
0×19	P	3	1,9	8	40	(683,1183)
0×20	S	10	0	1	(1.6,5)	1316
0×21	S	10	0	1	(1.6,5)	1316
0×22	S	10	0	1	(1.6,5)	1316
0×30	S	11	1	1	(1.6,5)	1316
0×31	S	11	1	1	(1.6,5)	1316
0×32	S	11	1	1	(1.6,5)	1316
$0 \times A$	P	7	5,9	4	4	17
$0 \times B$	P	2	12,9	2	2	1471
$0 \times C$	P	3	13,9	2	2	1471
$0 \times D$	P	2	13,9	2	2	1471
$0 \times E$	P	3	12,9	2	2	1471

Table A2 Total maximum load and VL count on transmitting switch ports

(p, S)	$(G_p, \lvert v_p \rvert)$
(0,0)	(20.4,4)
(1,0)	(20.4,4)
(2,1)	(12.3,7)
(3,2)	(12.3,7)
(4,3)	(0.14,1)
(5,4)	(0.14,1)
(8,0)	(0.04,2)
(9,0)	(49.9,24)
(12,1)	(12.2,2)
(13,2)	(12.2,2)
(14,0)	(18.4,8)
(14,1)	(12.5,4)
(15,3)	(12.3,5)
(16,0)	(18.4,8)
(16,2)	(12.5,4)
(17,4)	(12.3,5)

Acknowledgments

There has been no fund allocation for this research by any organization.

References

[1] Gaska, T., Watkin, C., and Chen, Y., "Integrated Modular Avionics—Past, Present, and Future," *IEEE Aerospace and Electronic Systems Magazine*, Vol. 30, No. 9, Sept. 2015, pp. 12–23. https://doi.org/10.1109/MAES.2015.150014

[2] Airlines Electronic Engineering Committee, "ARINC 664 P7-1: Aircraft Data Network Part 7 Avionics Full-Duplex Switched Ethernet Network," Vol. 9, No. 23, Aeronautical Radio, Inc., Maryland, 2009.

[3] "AFDX®/ARINC664P7 Tutorial," AIM, https://www.aim-online.com/products-overview/tutorials/afdx-arinc664p7-tutorial [retrieved 5 Aug. 2022].

[4] Tang, X., Li, Q., Lu, G., and Xiong, H., "A Revised Trajectory Approach for the Worst-Case Delay Analysis of an AFDX Network," *IEEE Access*, Vol. 7, Sept. 2019, pp. 142,564–142,573. https://doi.org/10.1109/ACCESS.2019.2943543

[5] OMNeT++ Discrete Event Simulator, OMNeT++, https://omnetpp.org [retrieved 2 April 2022].

[6] Varga, A., "Using the OMNeT++ Discrete Event Simulation System in Education," *IEEE Transactions on Education*, Vol. 42, No. 4, Nov. 1999, pp. 372–382. https://doi.org/10.1109/13.804564

[7] "AFDX—Avionics Full-Duplex Switched Ethernet Model for OMNeT ++," OMNeT++, https://github.com/omnetpp-models/afdx [retrieved 1 April 2022].

[8] "AFDX—Avionics Full-Duplex Switched Ethernet Model for OMNeT++," OMNeT++, https://omnetpp.org/download-items/Afdx.html [retrieved 5 Aug. 2022].

[9] Lauer, M., "Une Méthode Globale Pour La Vérification d'exigences Temps Réel : Application à l'Avionique Modulaire Intégrée," Institut National Polytechnique de Toulouse, 2012, https://theses.hal.science/tel-00714502 [retrieved 7 Aug. 2022].

[10] Safwat, N. E. D., Zekry, A., and Abouelatta, M., "Avionics Full-Duplex Switched Ethernet (AFDX): Modeling and Simulation," *32nd National Radio Science Conference (NRSC)*, Vol. 32, Inst. of Electrical and Electronics Engineers, March 2015, pp. 286–296. https://doi.org/10.1109/NRSC.2015.7117841

[11] Gwaltney, D. A., and Briscoe, J. M., "Comparison of Communication Architectures for Spacecraft Modular Avionics," NASA TM-2006-214431, June 2006, pp. 1–24, https://ntrs.nasa.gov/citations/20060050129.

[12] Felser, M., "Real-Time Ethernet - Industry Prospective," *Proceedings of the IEEE*, Vol. 93, No. 6, June 2005, pp. 1118–1129. https://doi.org/10.1109/JPROC.2005.849720

[13] Decotignie, J. D., "Ethernet-Based Real-Time and Industrial Communications," *Proceedings of the IEEE*, Vol. 93, No. 6, 2005, pp. 1102–1117. https://doi.org/10.1109/JPROC.2005.849721

[14] Sandic, M., Pavkovic, B., and Teslic, N., "TTEthernet Mixed-Critical Communication: Overview and Impact of Faulty Switches," *IEEE Consumer Electronics Magazine*, Vol. 9, No. 4, 2020, pp. 97–103. https://doi.org/10.1109/MCE.2020.2978224

[15] Zhao, L., He, F., Li, E., and Lu, J., "Comparison of Time Sensitive Networking (TSN) and TTEthernet," *IEEE/AIAA 37th Digital Avionics Systems Conference (DASC)*, 2018, pp. 1–7. https://doi.org/10.1109/DASC.2018.8569454

[16] Schuster, T., and Verma, D., "Networking Concepts Comparison for Avionics Architecture," *IEEE/AIAA 27th Digital Avionics Systems Conference*, 2008, pp. 1.D.1-1–1.D.1-11. https://doi.org/10.1109/DASC.2008.4702761

[17] Fuchs, C. M., "The Evolution of Avionics Networks from ARINC 429 to AFDX," *Seminar Aerospace Networks/Network Architectures and Services*, Vol. 65, Innovative Internet Technologies and Mobile Communications and Aerospace Networks (IITM & AN), Munich, Germany, Aug. 2012, pp. 65–76. https://doi.org/10.2313/NET-2012-08-1

[18] Boyer, M., and Fraboul, C., "Tightening End to End Delay Upper Bound for AFDX Network Calculus with Rate Latency FIFO Servers Using Network Calculus," *IEEE International Workshop on Factory Communication Systems*, May 2008, pp. 1–11. https://doi.org/101109/WFCS.2008.4638728

[19] Bauer, H., Scharbarg, J. L., and Fraboul, C., "Improving the Worst-Case Delay Analysis of an AFDX Network Using an Optimized Trajectory Approach," *IEEE Transactions on Industrial Informatics*, Vol. 6, No. 4, 2010, pp. 521–533. https://doi.org/10.1109/TII.2010.2055877

[20] Francés, F., Fraboul, C., and Grieu, J., "Using Network Calculus to Optimize the AFDX Network," *ERTS 2006: 3rd European Congress ERTS Embedded Real-Time Software*, Toulouse, France, 2006, pp. 1–8, https://oatao.univ-toulouse.fr/2159/.

[21] Charara, H., Scharbarg, J. L., Ermont, J., and Fraboul, C., "Methods for Bounding End-to-End Delays on an AFDX Network," *18th Euromicro Conference on Real-Time Systems (ECRTS'06)*, Dresden, Germany, Aug. 2006, pp. 193–202. https://doi.org/10.1109/ECRTS.2006.15

[22] Karthik, K., and Naga Vamsi, A. V., "Data Communication at Receiver End of AVIONICS System," *International Journal of Engineering Trends and Technology*, Vol. 21, No. 5, 2015, pp. 271–274. https://doi.org/10.14445/22315381/ijett-v21p249.

[23] Tian, Y., Ma, Z., and Zhou, S., "Analysis of AFDX Network Delay Based on NS2," *Journal of Physics: Conference Series*, Vol. 2026, No. 1, July 2021, Paper 012010. https://doi.org/10.1088/1742-6596/2026/1/012010

[24] Song, D., Zeng, X., Ding, L., and Hu, Q., "The Design and Implementation of the AFDX Network Simulation System," *International Conference on Multimedia Technology*, Vol. 2, Ningbo, China, Nov. 2010, pp. 1–4. https://doi.org/10.1109/ICMULT.2010.5629728

[25] Liu, X., Du, Z., and Lu, K., "Modeling and Simulation of Avionics Full Duplex Switched Ethernet(AFDX Network) Based on OPNET," *3rd Joint International Information Technology, Mechanical and Electronic Engineering Conference (JIMEC 2018)*, Vol. 3 (Atlantis Highlights in Engineering), Chongqing, China, Jan. 2018, pp. 307–310. https://doi.org/10.2991/jimec-18.2018.65

[26] Annighoefer, B., Ihle, H., and Thielecke, F., "An Easy-to-Use Real-Time AFDX Simulation Framework," *IEEE/AIAA 35th Digital Avionics Systems Conference (DASC)*, Sacramento, CA, Sept. 2016, pp. 1–9. https://doi.org/10.1109/DASC.2016.7778027

[27] Villegas, J., Fortes, S., Escaño, V., Baena, C., Colomer, B., and Barco, R, "Verification and Validation Framework for AFDX Avionics Networks," *IEEE Access*, Vol. 10, June 2022, pp. 66,743–66,756. https://doi.org/10.1109/ACCESS.2022.3184329

[28] Han, P., Zhai, Z., Nielsen, B., Nyman, U., and Kristjansen, M., "Schedulability Analysis of Distributed Multicore Avionics Systems with Uppaal," *Journal of Aerospace Information Systems*, Vol. 16, No. 11, 2020, pp. 473–499. https://doi.org/10.2514/1.I010715

[29] Weingärtner, E., Vom Lehn, H., and Wehrle, K., "Performance Comparison of Recent Network Simulators," *IEEE International Conference on Communication*, Dresden, Germany, June 2009, pp. 1–5. https://doi.org/10.1109/ICC.2009.5198657

[30] Zarrad, A., and Alsmadi, I., "Evaluating Network Test Scenarios for Network Simulators Systems," *International Journal of Distributed Sensor Networks*, Vol. 13, No. 10, 2017, pp. 1–17. https://doi.org/10.1177/1550147717738216

[31] "Simulation Models and Tools," OMNeT++, https://omnetpp.org/download/models-and-tools [retrieved 5 Aug. 2022].

[32] Steinbach, T., Dieumo Kenfack, H., Korf, F., and Schmidt, T., "An Extension of the OMNeT++ INET Framework for Simulating Real-Time Ethernet with High Accuracy," *4th International ICST Conference On Simulation Tools and Techniques*, March 2011, pp. 375–382.
https://doi.org/10.4108/icst.simutools.2011.245510

[33] Yang, Q., Lu, H., and Tu, X., "Simulation and Experiment of AFDX Network Based on OMNeT++," *Chinese Automaton Congress (CAC)*, Nov. 2020.
https://doi.org/10.1109/CAC51589.2020.9326633

[34] Rejeb, N., Ben Salem, A. K., and Ben Saoud, S., "AFDX Simulation Based on TTEthernet Model Under OMNeT++," *International Conference on Advanced Systems and Electric Technologies (IC_ASET)*, Inst. of Electrical and Electronics Engineers, New York, Jan. 2017, pp. 423–429.
https://doi.org/10.1109/ASET.2017.7983731

[35] Kultur, O. R., and Bilge, H. S., "Comparative Analysis of Next Generation Aircraft Data Networks," *IEEE International Conference on Smart Technologies (EUROCON 2021)*, Inst. of Electrical and Electronics Engineers, New York, July 2021, pp. 317–320.
https://doi.org/10.1109/eurocon52738.2021.9535577

[36] Atik, E., "A New Fault Tolerant Real-Time Ethernet Protocol: Desing and Evaluation," Master's Thesis, Electrical and Electronics Engineering Dept., Middle East Technical Univ., Ankara, Turkey, 2021.

[37] "Badapplexx/AFDX," GitHub, Inc., https://github.com/badapplexx/AFDX [retrieved 2 April 2022].

[38] "GitHub—Badapplexx/ANCAT: AFDX Network Analysis Tool," GitHub, Inc., https://github.com/badapplexx/ANCAT [retrieved 10 Dec. 2023].

[39] Harchol-Balter, M., *Performance Modeling and Design of Computer Systems: Queueing Theory in Action*, Cambridge Univ. Press, Cambridge, England, U.K., 2013, pp. 95–111.

[40] "OPNET Network Simulator—Opnet Projects," OPNET, https://opnetprojects.com/opnet-network-simulator/ [retrieved 10 Dec. 2023].

[41] Siddaraju, N., and Rao, K., "Multiplexing the Elementary Streams of H.264 Video and MPEG4 HE AAC v2 Audio, De-Multiplexing and Achieving Lip Synchronization," *American Journal of Signal Processing*, Vol. 2, No. 3, 2012, pp. 52–59.
https://doi.org/10.5923/j.ajsp.20120203.03

C. Torens
Associate Editor