The Evolution of Avionics Networks From ARINC 429 to AFDX

Christian M. Fuchs
Advisors: Stefan Schneele, Alexander Klein
Seminar Aerospace Networks SS2012
Chair for Network Architectures and Services
Faculty of Informatics, Technical University of Munich
Email: christian.fuchs@tum.de

ABSTRACT
Signaling and inter-system communication in avionics have been crucial topics ever since electronic devices were first used in aerospace systems. To deal with the challenges introduced by the widespread use of general purpose computing in commercial avionics, standards like ARINC 419 and later on 429 were published and adopted by the industry. While in industrial use, 429 has been adapted and extended very little since the standard was formulated in the late 1970s. 429 today cannot meet challenges and new requirements generated by the use of Integrated Modular Avionics and flexible system design. AFDX combines proven safety and availability functionality with modern Ethernet technology to be able to handle today's requirements. This paper outlines two of the most important avionics network architectures and aims at depicting the evolution of networking concepts and requirements over the course of the past 30 years. It mainly focuses on ARINC 429 and AFDX, the most prominent current and past standards, but also covers two other interesting past protocols.

Keywords
AFDX, ARINC 664, ARINC 429, Ethernet, MIL-STD-1553, avionics, fault tolerance, security, safety

1. INTRODUCTION
Signaling and inter-system communication in avionics have been a crucial topic ever since electronic devices were first used in aircraft. Initially, simple sensory feedback and components like radar and engines needed to be interconnected with cockpit controls. As time progressed, more and more systems which produce and consume data were introduced in avionics, at some point becoming crucial for even the most essential tasks, such as steering and later fly-by-wire. To deal with these challenges in commercial avionics, standards like ARINC 419 (and later on 429) were drafted and adopted not just by individual corporations, but collectively by almost the entire industry [1, 2].

Today, ARINC 429 can be found in most active and retired aircraft series. While it is well-established in the industry, it has been adapted and extended little since the initial specifications were formulated in the late 1970s. In contrast to avionics standards, multiple technological revolutions have happened in the computer industry at a fast pace. Networking of computers aboard aircraft may have been unthinkable in 1970, while modern aircraft without any networked computers are very uncommon. Legacy avionics communication standards still reflect past views on computing [1, 3].

Ultimately, a modern networking architecture for avionics use should offer a maximum of safety, redundancy and security, as well as apply failsafe defaults. The resulting infrastructure should be efficiently maintainable, flexible and offer a solid foundation for software development. More recent standards reflect these demands, though few saw broader use across the industry [4].

In contrast to the Internet, security and cost efficiency are not the key objectives in avionics; rather safety is. However, most modern networking standards are aimed at achieving traditional PC-world security objectives and only indirectly address safety requirements (by fulfilling traditional security objectives) [5, 6].

In ARINC 664 Part 7, also referred to as AFDX, standard Ethernet technology is extended and design objectives are built around safety.

Two of the most important network architectures in the avionics industry are outlined in this paper, and we aim at depicting the evolution of networking concepts and requirements over the course of the past 30 years. It mainly is focused on the most prominent current and past standards, ARINC 429 and 664, but also covers two other significant standards (MIL-STD-1553 and ARINC 629). These standards introduced important features into aerospace networking design and are used as intermediate steps in this paper even though AFDX evolved independently.

In this paper, a deeper understanding of Ethernet is assumed; the reader should be familiar with redundancy and failover concepts, as well as information-security. The OSI layer model is used throughout this paper, even though it is not used within the cited avionics standards. When referring to layer 2 (L2) frames, Ethernet or AFDX frames at the data link layer are meant, while L3 and L4 refer to data structures used in the respective protocol at the network and transport layers.

Within the next section, the most widespread standard, ARINC 429, is explained in detail. In Section 3, the transition from federated network architectures, such as 429 to modern *Integrated Modular Avionics*, is depicted. Then, an analy-

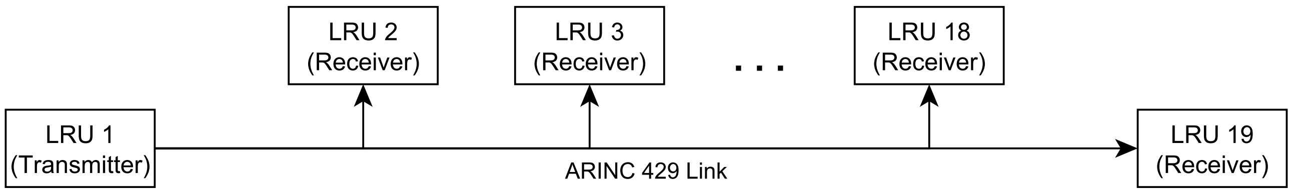


Figure 1: An ARINC 429 layout with just one transmitting LRU and up to 19 recipient. [4]

sis of the reference operating system proposed in ARINC 653 for use with integrated architectures is conducted. In Section 4, ARINC 629 and Mil-Std-1553, two more recent networking standards are briefly introduced. Section 5 is focused on the networking standard AFDX. The emphasis is on the enhancements to Ethernet needed to comply with the requirements of avionics applications. The final chapter is dedicated to summarizing the advantages and disadvantages of the main two named architectures.

2. ARINC 429

In the following section, the ARINC standard 429 will be depicted in detail. We will especially focus on its architectural principles, history and capabilities. Subsequently, the limitations imposed on networks design will be outlined.

2.1 Basic Topology

ARINC 429 (in full, *Mark 33 Digital Information Transfer System*), implements serial line communication and was one of the first standards specifically targeted at avionics applications. ARINC 429's predecessor, the commercial standard 419 [2], was first published in 1966, with 429 being based on (from a 1978 viewpoint) more recent technology [1].

429 specifies direct wiring of LRUs using a serial *twisted shielded pair*-based interface that can connect peers which are up to about 90 meters apart. Unlike in modern networking protocols, this is a signaling standard; thus the sender always sends to the line, and the recipients always read from it. If no data is available for sending, the line is set to zero-voltage. Even though *twisted shielded pair* cabling is used, all lines are simplex connections populated by one single sending station and multiple recipients (up to 19), as shown in Figure 1 [1].

The bus, also referred to as a *multi-drop bus*, can operate at *low* or *high speed. Low speed* uses a variable clock rate and a nominal throughput of 12-14 kbps, while the *high speed* mode requires a fixed clock rate and allows 100 kbps.

In ARINC 429 terminology, each chunk of data transmitted over a link is called a *word*; the word format is defined in the next section. Two types of words exist: data words and message control words. Messages consist of multiple words of data, which are then called records [4].

Link control messages are used in a similar way as in modern networking stacks. If a listening LRU is ready to receive data, a *"Request to send"* message will be transmitted via its dedicated sending link. *"Clear to send"* is used for the opposite. *"data follows"*, *"data received OK"*, *"data received not OK"* and *"synchronization lost"* can be used by the sender/recipient for further interaction. Each *message* is started by the message control word *"data follows"*, followed by up to 126 data words [1].

In contrast to today's networking standards, as 429 defines a unidirectional and simplex bus, recipient LRUs may not send messages to the bus they are listening to. This includes messages related to link control.

A station may be attached to multiple buses and operates as either sender or recipient, thus hierarchical layouts are possible. However, bidirectional information exchange between systems is at least required for message control and acknowledgment [1]. In this case, recipient LRUs respond via a secondary interface, on which the positions of sender and recipient are interchanged. As only one single station participating in an ARINC 429 link can occupy the sender role, one back-channel for each LRU is required [4].

2.2 Word Format and Data Types

Multiple data encoding formats which are targeted at different usage scenarios in avionics are defined: *Binary*, *Binary Coded Decimal* (BCD, see Figure 3), *Discrete Data*, File transfer using the ISO 646 character set, and *Maintenance Data and Acknowledgment* [1].

The structure of ARINC 429 words does not conform to modern communication standards; it is non-byte-aligned, but optimized for keeping down latency.

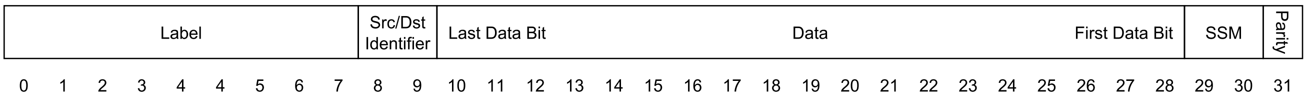


Figure 2: The data format of an ARINC 429 word. [1]

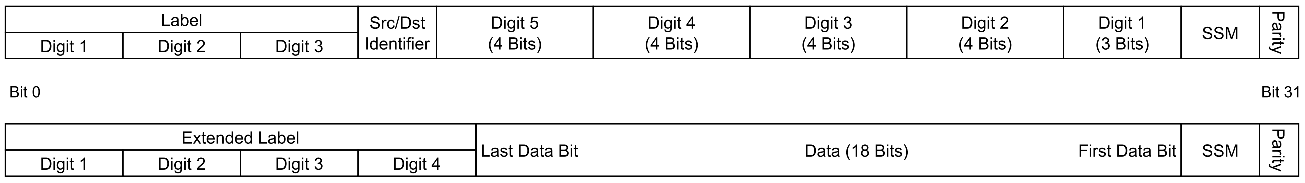


Figure 3: Two 429 words in *binary coded decimal* format(above) and *binary* format. The binary word has an extended label (4 digits/3 bits each), whereas the BCD word in this example uses a regular 3-digit label. [1]

Common fields in all word formats are:

- *label* (8 bit),
- *source* and *destination identifiers* (optional, 2 bit),
- the *sign/status matrix* (2 bit), and
- *data* (19 bit) field.

Words are terminated by a single parity bit. The total size of all words is 32 bits as depicted in Figure 2.

The small message size results in very low latency, minimizing delays during processing and guaranteeing timing, as no transport side queuing or rescheduling of traffic can happen. The message size is one of the cornerstones for achieving safety, resilience and reliability [1]. No other communication standard currently in use in avionics offers the same level of responsiveness, which makes it an interesting choice for applications where a delay of a few milliseconds may be too much [7].

Depending on the chosen data format, the *Sign/Status Matrix* flags have predefined meaning; for example, if the BCD format is used, setting both SSM bits to zero means *North, East, Right, To* (directional), *Above or Plus*. Other bit-patterns and data encoding variants have different predefined meanings, which LRUs are supposed to support to retain compatibility [1].

A word's label is used as frame header, holding information on the encoding format and three octal numbers, which can be used by LRUs to select matching messages upon receipt. The label may be extended by three bits, which then serve as fourth label digit. Label digit codes are predefined in the standard and have fixed meanings, too. The label is sent high-order-bit first, whereas the rest of the message is sent least-significant-bit first [1].

2.3 Limitations

Due to the simplistic layout of 429 links, each individual connection is a physical cable, which allows for easy testing, as either a LRU or the line itself may be faulty. This also poses a severe challenge when designing systems with dense interlinking.

As depicted in Figure 4, even an environment with few stations present may become very complex once a certain degree of interaction is needed [1]. In modern commercial aircraft, interlinking between a multitude of systems as well as extreme cabling overhead may occur, imposing severe limitations on network design as well as impacting the overall weight of such a craft [7].

Bit-error correction via symmetric or cryptographic checksum algorithms is not designed to happen within ARINC 429 but instead needs to be implemented at application level. In fact, all data processing must be handled by each LRU's software directly. There is no uniform, device-independent 429 software stack [1].

Overall, custom proprietary (and thus, expensive) hardware is required to implement an ARINC 429 setup, which is common in the aerospace industry. Almost no consumer-off-the-shelf hardware is available, with the exception of cabling. However, it should be noted, separate aviation-standards apply to cabling. Software development is problematic too, as no modern day networking protocols can be used, and development is done for highly specialized hardware. Retrofitting of older aircraft with new technology may thus be costly.

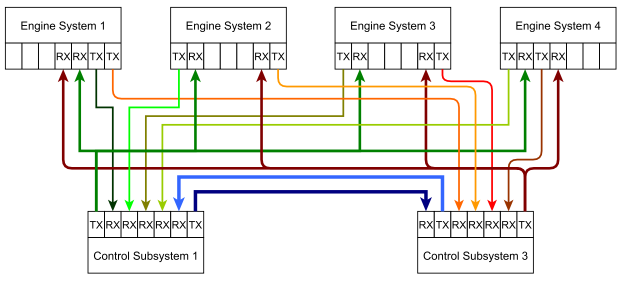


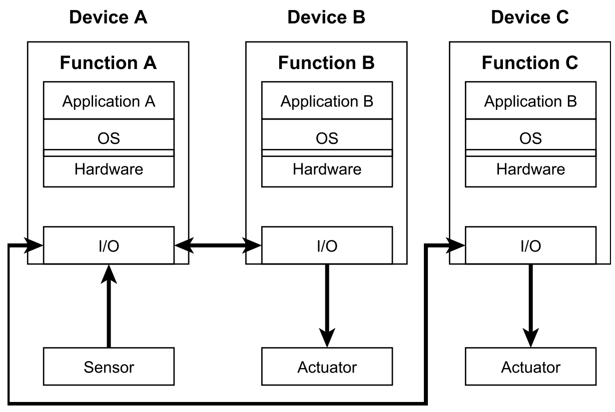
Figure 4: An ARINC 429 network containing just two control subsystems and four data-consuming components. [7]

3. INTEGRATED MODULAR AVIONICS

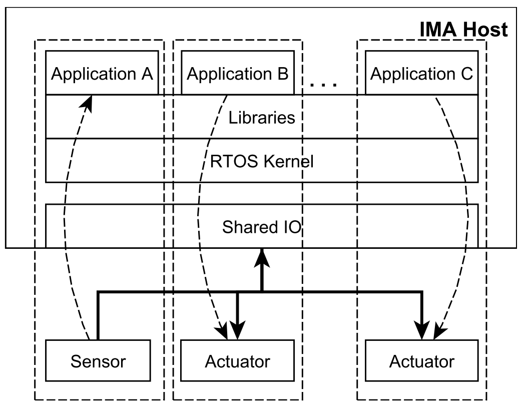
As has been outlined in the previous section, federated architectures severely restrict scalability and flexibility of computerized environments with dense interlinking. Thus,we will take a look at the modern day alternative to federated avionics in this section. First Integrated Modular Avionics will be introduced, followed by a brief analysis of ARINC 653.

3.1 Towards Integrated Modular Avionics

ARINC 429 allows the implementation of federated network architectures. It does not distinguish between hardware and software, but rather between devices specially built for a single purpose (e.g. polling a sensor, transmitting radar signals, issuing steering commands, etc). LRUs are thus unique to a certain degree and also need to be certified as a whole. As there is no distinction between hardware and software,



(a) Federated Network Layout



(b) Integrated Modular Avionics

Figure 5: A comparison of network architectures. [8, 9]

re-certification needs to take place even if only software was updated or replaced [10].

As software may require additional system resources in the future (more RAM, disk space, CPU-power, ...), devices need to be constructed with appropriate reserves. Components in federated networks (usually) cannot share resources and reserves need to be defined on a per-system level, network-wide resulting in a high degree of idle capability, extra weight and extra cost. The overall result is a proprietary network with potentially extreme cabling overhead containing a multitude of different line replaceable units or modules fulfilling unique roles [11].

Due to the described limitations, the aerospace industry has begun to move away from federated architectures in favor of *Integrated Modular Avionics* (IMA) [9, 12]. Most importantly, IMA does discriminate between software and hardware and defines an abstraction layer between physical hardware and software-implemented functionality. However, IMA still allows the use of monolithic, unique LRUs, but encourages the homogenization of on-board hardware [10].

Multiple functional packages implemented by different software may be executed on a single IMA-host, as depicted in Figure 5. As such, applications must be isolated from each other, I/O ports and system resources need to be provided to the correct applications. As resources on an IMA-host are shared and available to all applications, reserves can be calculated collectively for all run applications [13].

3.2 ARINC 653

As described in later sections, timing and timing constraints are critical in avionics and networks design. These restrictions cannot be met by regular operating systems; hence, the use of a real time operating system (RTOS) is mandatory in IMA [12]. A reference implementation of such an operating system is specified in the ARINC standard 653, including a matching API and communication concepts for use on an IMA-host. The standard is categorized in different parts, covering essential and extra services that can or must be provided by the OS, as well as guidelines for testing [13].

In contrast to the operating systems used in most of the computer industry, 653's API *APEX* allows applications to remain completely independent of the underlying system's architecture, as shown in Figure 7. It grants applications access to hardware, allows file handling through a dedicated core service and permits access to the virtual networks backplane. Applications are assigned queuing ports, which are FIFO buffers for packets, and sampling ports, which are single-packet buffers to be overwritten upon each iteration. In conjunction with *APEX*, communication ports allow time-deterministic networking [13].

In 653, the core components of the RTOS are defined; many components are identical or very similar to those in use in common non-avionics real time operating systems and will thus not be described further. Others, like partition management and the *health monitor*, are uncommon. The *health monitor* provides hardware, software and network status information to the OS as well as the partitions. Thus, both the RTOS' core services as well as applications must support health monitoring functions [11].

Feedback on faults related to system services allows the *health monitor* to initiate countermeasures in case of failure (e.g. memory relocation, starting of service routines, etc). In case application partitions receive relevant information on issues, they can also take countermeasures on their own by selecting a different path to a sensor in the network or switching to fail-over functionality [11].

Health monitoring requires classification and categorization based on information provided by the application's author or manufacturer of the individual device. Thus, errors relevant for health monitoring are handled and also detected at different levels on an IMA host. Information on all of these error handling strategies are collected in recovery strategy tables at a per-partition level and at the system level. The tables themselves are to be maintained by the aircraft designer.

Isolation was one of the key design objectives in 653 and IMA, as they must allow side-effect-free executions of applications. Applications only need to properly make use of *APEX* and can be completely isolated from each other or can be permitted IPC [11]. *APEX* and the underlying system libraries are usually considered as one single functional block. Each application is run inside an isolated partition[1]. The partition management services of the RTOS are responsible for assigning priorities and timing constraints for the individual application partitions [13].

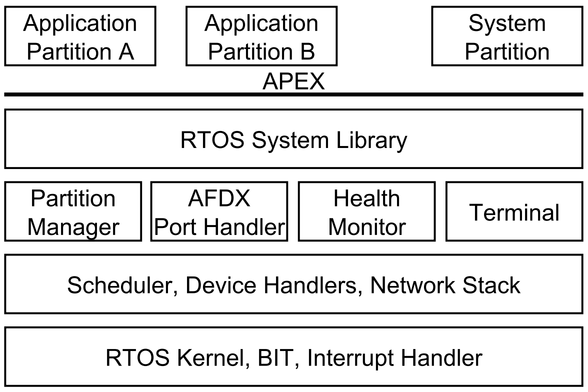


Figure 7: The functional layout of an operating system as specified in ARINC 653. [13, 14]

Therefore, if ARINC 653 is implemented, hardware and software can be certified incrementally. If software is replaced or updated, only the individual application or the OS is required to be re-certified. The underlying hardware remains unaffected and does not require additional steps to be taken beyond contingency planning and mitigating the need for possible extra systems resources [12, 13].

Strict guidelines are imposed on application design as well. In 653, among other guidelines, the way application software and core services of the OS are supposed to store and handle files is specified (i.e. configuration files are to be stored as XML files). *APEX* compliant applications can either run persistently or transiently, and thus can be started, stopped or moved to other IMA-hosts if it is permitted by the systems and networks configuration. IMA thus requires abstraction of networking hardware to satisfy software requirements.

[1]653 partitioning is not to be confused with hardware partitioning commonly in use in commercial high performance computing.

4. LEGACY NETWORKING STANDARDS

Strictly federated ARINC 429 networks do not offer the logical abstraction required for deploying integrated modular avionics; a shared medium and logical abstraction of interfaces is needed. In the following section, legacy network architectures offering such abstractions will be described briefly. First we will take a look at ARINC 629; while it is used rarely in avionics, it introduced a very important concept for gaining determinism in a shared medium network. Then, we will investigate into one of the most widely deployed military networking standards, Mil-Std-1553b.

4.1 ARINC 629

ARINC 629, also known as *Digital Autonomous Terminal Access Communication*, was developed jointly by Boeing and NASA to overcome limitations imposed by 429, and was later handed over to ARINC. Though more modern than previous standards, it never saw wider use in the industry. It was almost exclusively used in the Boeing 777 and even this aircraft contained additional 429 backup infrastructure. According to 629, the deployment of a triplex-bus layout, similar to the one specified by 10BASE2/10BASE5, was foreseen. It also implements CSMA/CD as used in Ethernet [16]; thus, collisions may occur on the bus. The original standard defined a clock-speed of 2 MHz, which can be raised as technology advances, and the bus was built upon twisted pair cabling from the very beginning. Like ARINC 429 it uses 32 bit data words, but the data element is 2-byte-aligned [4].

While 429-based architectures support only direct peer-to-peer communication, 629 mainly uses directed unicast and allows broadcast on a shared medium. It was an early attempt to increase flexibility and reduce cabling effort. Flexibility and the added complexity resulted in increased signal latency and reduced determinism, while the concept in general still lacks support for COTS-hardware. Thus, bus access by multiple stations in a 629 network may happen in random order, depending on which system sends first.

Parts of the basic protocol in 629 have found their way into modern standards, especially the use of predefined gaps between transmissions to prevent stall or packet loss to add determinism. As depicted in Figure 6, the *Synchronization Gap* is a random per-frame interval, whereas the terminal gap is distinct for each terminal.

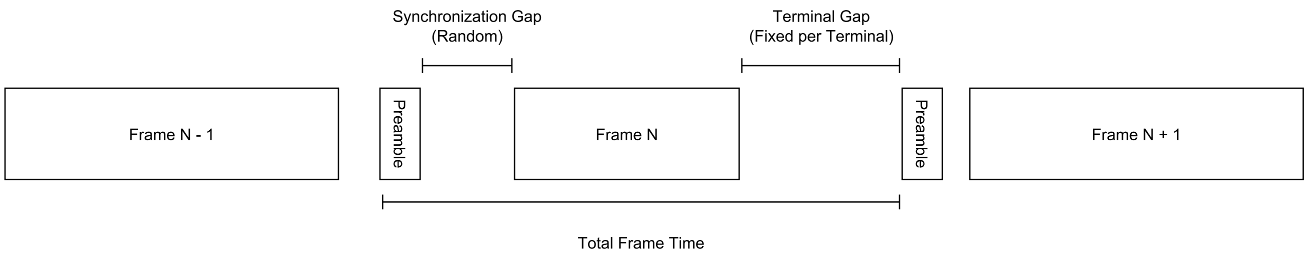

Synchronization Gap (Random)
Terminal Gap (Fixed per Terminal)
Frame N - 1
Preamble
Frame N
Preamble
Frame N + 1
Total Frame Time

Figure 6: Frame-timing as specified in ARINC 629. [4]

4.2 MIL-STD-1553b

MIL-STD-1553b, the *Aircraft Internal Time Division Command/Response Multiplex Databus*, is widely used in military aircraft (e.g. Airbus's A400M) and even the International Space Station [13]. It is an interesting approach that also helped bridge the gap between basic signaling and modern networking standards like AFDX. The standard was evolved from the initial legacy data rate of 1 Mbps to the *extended* and *hyper* variants, which use newer hardware to offer 120 and 200 Mbps respectively [7]. In contrast to other bus based standards, MIL-STD-1553b specifies a logical star on top of the physical bus topology. This topology is called *robust physical layer* and uses *triaxial cabling* [15].

In the standard the role of a *bus controller* is defined. This device is responsible for initiating all communication between *subsystems*(peers) on the bus through a command-response protocol. In case the *bus controller* fails, another *remote terminal* can take over this role for the time being. To ensure fail-over, multiple redundant bus instances operate in parallel (see Figure 8), while each bus still only allows half-duplex communication [3].

Subsystems sending data via the bus are not directly connected, but instead use individual *remote terminals* to access the network. All communication on the bus is supervised by the *bus monitor*, which may also perform logging of parts or all the communication. Usually, transmissions are unicast, and are thus only exchanged between two remote-terminals. Broadcast is supported by design but discouraged [3].

In 1553, data words are just 20 bits long, but there is less protocol overhead than in other standards, as all communication is directed by the *bus controller*, and peers only execute commands they were given (e.g. read from bus, send to terminal).

To save on bit-space, three word formats exist:

- command word (sent by the *bus controller*),
- status word (response from LRUs to *bus controller*),
- and data word format.

While the standard is still incompatible with modern day networking protocols and does not adhere to the OSI layer model, it allows an LRU to emulate logical links, which is required for integrated system architectures [17].

5. AVIONICS FULL DUPLEX PACKET EXCHANGE

In this final section, we will take an in depth look at *Avionics Full-DupleX Ethernet switching* (AFDX). We will analyze how it was designed to extend standard Ethernet to meet todays requirements in an aerospace environment. Afterwards, the key elements of an AFDX network and the changes necessary to upper layer protocols will be described.

5.1 A brief history of ARINC 664

As an evolved standard, 429 had many limitations, but it is a proven and commonly used protocol. As time progressed and technology advanced, more bandwidth, more flexible topologies and new challenges like *Integrated Modular Avionics* (IMA, see Section 3) [13,17] emerged and were beyond ARINC 429's capabilities [12,18].

ARINC 664 (Part VII) was initially developed by the EADS Airbus division as *Avionics Full-DupleX Ethernet switching* (AFDX). Though previous aircraft already deployed fully electronic fly-by-wire systems, wiring using previous standards could no longer meet the requirements of modern day state-of-the-art aircraft. In the case of AFDX, the Airbus A380 prompted for a new technological base to be implemented; thus, AFDX was created. Later on, Airbus' AFDX was transformed into the actual ARINC standard [19]. Figure 9 shows a simple AFDX-based Network.

5.2 From Ethernet to AFDX

5.2.1 Architectural Changes

Ethernet has been in use for decades outside of the aerospace industry and proved to be a robust, inexpensive, extensible and flexible technology. However, it cannot offer essential functionality required for high availability and reliability. Thus, it is not directly suitable for avionics. 664 offers modern day transfer rates, while building on top of the previously much-loathed Ethernet standard 802.3 [16]. AFDX inherits parts of the MIL-STD-1553 terminology and overall setup. Devices transmitting data via the network are called *subsystems*, which are attached to the network via *end systems*. The full-duplex network itself is called *AFDX Interconnect*; in Ethernet terms, this includes all passive, physical parts of the network, but not switches and other active devices [7].

The most prominent hindrance for using Ethernet networking in avionics is Ethernet's non-determinism. For regular computer networks, packet loss and timeouts are a common

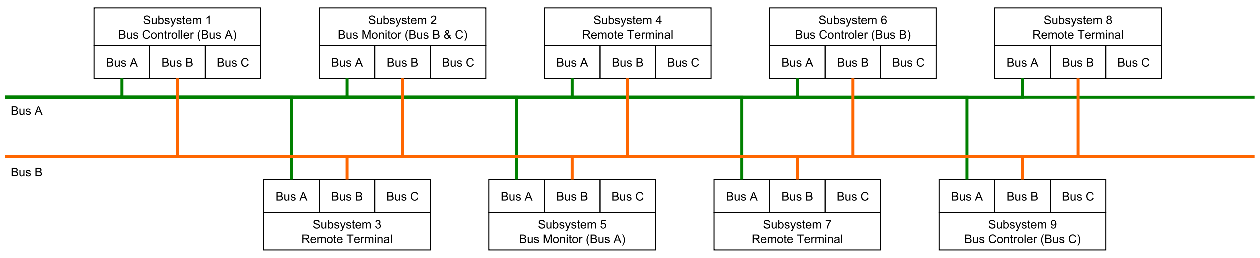


Figure 8: A single redundant MIL-STD-1553 bus network with hardware and device roles predefined for provisioning a second fail-over bus C. [3,15]

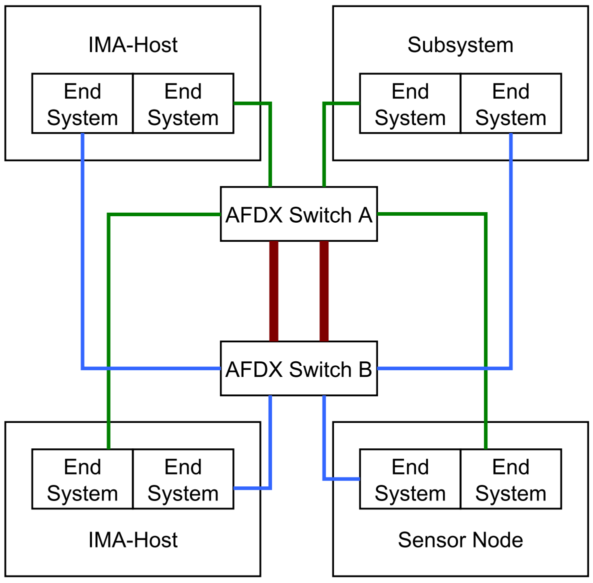


Figure 9: An example of an AFDX based network. Each subsystem is attached physically to the network by two *end systems*. [19]

issue. Upper layers, such as a station's operating system or applications, are supposed to handle these issues by design. If a message is lost or corrupted during transmission, it will simply be resent or its loss fully mitigated. When sending data on a non micro-segmented network, collisions may occur in each segment, forcing all stations involved in the collision to resend. Transmission of packets is retried after a random time interval by whichever station starts first. Again, a collision may occur which may lead to next to indefinite repeating, and this may subsequently result in a jammed bus [19].

Another variable factor of Ethernet networking and subsequently ARINC 664, are switches/bridges. While they add flexibility to networking, additional non-determinism is introduced, as frames may be reordered or manipulated in transit. Switches offer micro-segmentation of network segments, but in turn also increase the number of hops a frame takes from source to destination. Thus, latency is increased and timing behavior may vary if frames move along multiple paths [19]. In highly congested setups, switches may even drop packets on purpose if buffer limits have been reached[2].

In Ethernet, collisions are handled via CSMA/CD, but upper layers may encounter packet loss. There, protocols (e.g. TCP, SCTP, etc) in the operating system's network stack have to deal with packet loss [13]. However, this is not a viable solution in safety-critical environments. Certain applications require bandwidth guarantees, while others may demand timing behavior to remain within strict boundaries. Neither can be offered by Ethernet. No *hard* quality of service guarantees are available in vanilla Ethernet, and soft scheduling is only offered through protocol extensions such as Ethernet-QOS IEEE 802.1p. The same applies to bandwidth allocation, which can not be guaranteed in Ethernet on a per-flow level, but is implemented using various different algorithms. While there are several proprietary approaches for making Ethernet usable in real-time environments, none of these standards is directly usable in avionics [20, 21]. Thus, the new standard required determinism to make it usable in avionics [19].

5.2.2 Virtual Links

Ethernet is independent of physical connections and allows logical endpoints to be defined. Multiple physical or virtual devices may thus share one link, supporting virtual subsystems or virtual machines in IMA [12, 13, 18]. Multiple applications or devices may require different timing characteristics or a fixed minimal amount of bandwidth [19].

Virtual point-to-point connections implement the same concept as used in ARINC 429. In contrast to 429, they do not exist physically, but as logical links. They are implemented as *Virtual Links* (VL) on top of the AFDX Ethernet layer. An example of virtual channels is given in Figure 10. To a certain degree, VLs are quite similar to VLAN tagging as defined in IEEE 802.1Q [22], but offer additional information in addition to network isolation. Each virtual channel has three properties besides its channel ID: the Bandwidth Allocation Gap, the maximum L2 frame size, called LMAX or Smax, and a bandwidth limit [4].

LMIN and LMAX are used to set a predefined smallest and largest common Ethernet frame size along the path a packet

[2]In a properly laid out AFDX network, buffer overruns should never actually occur. The network parameters are configured based on values calculated during the planning phase of an aircraft using a mathematical framework.

ASN Channels	**Sensor Rate Groups (BAG value)**			
	1 msec	8 msec	32 msec	128 msec
Sensor Pod 1 (VL ID#)	0 - 7	8 - 15	16 - 23	24 - 31
Sensor Pod 2 (VL ID#)	32 - 39	40 - 47	48 - 55	56 - 63
Sensor Pod 3 (VL ID#)	64 - 71	72 - 79	80 - 87	88 - 95
Sensor Pod 4 (VL ID#)	96 - 103	104 - 111	112 - 119	120 - 127
Data Size	64 bytes	128 bytes	256 bytes	512 bytes
Busy Time/Channel	0.84 µsec	1.48 µsec	2.76 µsec	5.32 µsec
Total Time	26.88 µsec	47.36 µsec	88.32 µsec	170.24 µsec
Netto Bandwidth	**21.504%**	**4.736%**	**2.208%**	**1.064%**

Table 1: Impact of the *bandwidth allocation gap* on virtual link performance. [13, p. 11]

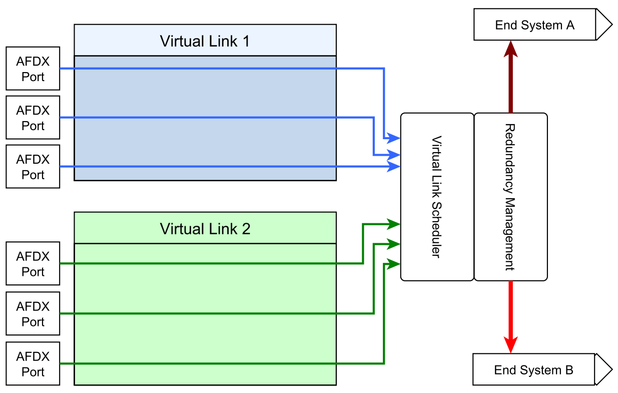


Figure 10: AFDX ports are abound to *virtual links*, characteristics specified by ports as well as VLs result in the actual VL parameters. [4]

may take, removing the need for IP-Packet fragmentation and similar mechanisms, thus removing yet another source of non-determinism from Ethernet [4].

Once the buffers present in switches or end stations are filled up, physical links can become congested and packets would be lost or delayed in transition. In safety-critical environments this is unacceptable; thus, the so called *Bandwidth Allocation Gap* (BAG) was defined. The BAG is defined in software for each VL, setting the delay (1-128 milliseconds) between sending the VL's frame and the next. In congested networking scenarios, frames will be sent within the time-window given by the BAG [19].

By setting LMAX and a reasonable BAG, bandwidth is segmented and can thus be guaranteed for each VL individually. The downside of introducing timesharing is a considerable loss of bandwidth if multiple links with different properties are defined. Table 1 reflects this loss of throughput depending on the network's parameters. The data was taken from benchmarking conducted by NASA in [13, p. 11] based on a 100 Mbps ancillary sensor network configuration. Even if little data is sent, one single VL with high BAG (long wait-time between frames) and low LMAX (small supported frame size) can drastically reduce overall network throughput [7].

Virtual Links are designated using so called *Virtual Link Identifiers* (VLID). The VLID replaces MAC-address based delivery, occupying the bits normally used for the destination MAC. To retain compatibility with Ethernet, AFDX splits the destination-MAC field into multiple parts: the initial bits are set to reflect a locally administered MAC-address (site-local), the final 16 bits store the VLID [19].

Only one *subsystem* may send data using a given VLID, thus *Virtual Links* are again unidirectional. As in ARINC 429, a *subsystem* can assume different roles in multiple VLs using different ports (see below), and multiple recipients may participate in a *Virtual Link*. Subsystems are not explicitly addressed, as in common Ethernet where MAC addresses are used, but the meaning of a *Virtual Links* identifier is defined and enforced by the network configuration [23]. However, parts of the original MAC address data area are designated user specifiable.

To make use of AFDX's capabilities, traditional socket programming is insufficient. Thus, a layer of abstraction was added: the communication ports. Communication ports - that is *sampling* and *queuing ports* - are accessed through a dedicated networking API (see ARINC 653) by a *subsystem*. Ports are assigned to *Virtual Links* and used to exchange data between *subsystems*; *end systems* deliver messages to one of the *subsystem's* ports, multiple ports at a *subsystem* can be members of a VL, but each port may only be attached to a single VL.

Sampling ports have dedicated buffer-spaces in which one single message can be read and stored. If a new message arrives, previous data will be overwritten. A *queuing port's* buffer may contain up to a fixed number of messages that are stored in a FIFO queue; upon reading the oldest message, it is removed from the queue. Handler services for communication ports need to be provided according to the ARINC 653 specifications [12]. BAGs and LMAX of a VL should be set accordingly to the collective requirements of all ports participating in a link [19].

Scheduling is performed on a per-port and per-link level at each *end system* independently based on the previously named properties. End systems ensure *Virtual Links* do not exceed their bandwidth limits, when multiplexing transmissions from different ports and *Virtual Links*. Also, jitter must be kept within fixed boundaries (specified by the standard), as frame-transmissions are offset by jitter within the BAG [19]. *Virtual Links* are scheduled before redundancy is applied to transmissions.

Subsequently, data passing through a Virtual Link will always take the same path in an active AFDX network, as they are predefined on the AFDX-switches along the route. As reconfiguration of AFDX devices does not happen at runtime, this path will persist until the relevant devices in the network are reinitialized. This implies timing behavior at runtime will remain constant for each frame (as defined by the BAG), even if individual devices along the path fail. Single failures will have no impact on the overall setup due to the provided redundancy. More on AFDX-switching will be discussed in Section 5.2.4

5.2.3 Redundancy

High availability environments also require redundancy on the bus as well as within stations. Again, Ethernet does not offer any sort of fail-over by default, however, optional link aggregation as defined in IEEE 802.1AX [24] can offer such functionality. 664 by design specifies sophisticated redundancy concepts for end stations as well as cabling by providing two dedicated networks (network A and B). After scheduling of Ethernet frames, redundancy is introduced. Each AFDX *subsystem* has two interfaces called *end systems*. Redundancy is added transparently by sending each frame via both *end systems*, applying the frame sequence number (see Figure 10). Assuming no transmission errors occurred, one duplicate will arrive at the destination for each frame transmitted [19].

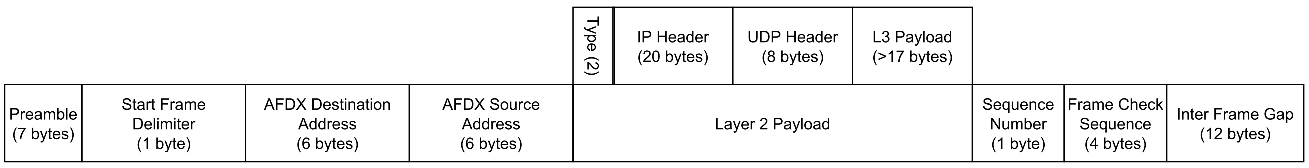

Type (2)
IP Header (20 bytes)
UDP Header (8 bytes)
L3 Payload (>17 bytes)
Preamble (7 bytes)
Start Frame Delimiter (1 byte)
AFDX Destination Address (6 bytes)
AFDX Source Address (6 bytes)
Layer 2 Payload
Sequence Number (1 byte)
Frame Check Sequence (4 bytes)
Inter Frame Gap (12 bytes)

Figure 11: The layout of an ARINC 664 frame is almost identical to an standard Ethernet frame and can be transported via COTS Ethernet hardware. [19]

Duplicate frames must be detected and discarded. To do so, a single byte sequence counter was appended to each OSI layer 3 packet, turning the resulting Ethernet frame into an AFDX Frame, as depicted in Figure 11. The AFDX sequence number is added as a trailer to the Layer 2 payload. For each packet sent via a *Virtual Link*, the counter is incremented. The first frame with a valid checksum to arrive at a *subsystem* is used for processing, subsequent frames with a duplicate sequence number can then be identified and discarded [4].

In case a of failure on the *AFDX Interconnect*, software and switches can be made aware of this issue and may divert traffic along other, predefined fail-over paths within the same network without having to rely solely on the independent network redundancy [11].

AFDX currently supports line speeds of up to 1 Gbps (see Table 2 created by [7]), but may support faster transports as technology advances. End systems are connected to switches directly and use full-duplex links [4]. As defined in the standard and due to the use of fully-switched Ethernet, only two stations exist in each AFDX Layer 2 segment: an *end system* and a switch, two switches, or two *end systems*. Collision domains are thus minimized. Subsequently frame collisions that need to be resolved via CSMA/CD on a single segment essentially should not happen [25].

5.2.4 AFDX Switches

Most features AFDX consists of can also be implemented using regular Ethernet hardware, if special AFDX-stack implementations are run. While purely software-based implementations exist [23], these solutions can not guarantee determinism. They can not keep jitter within boundaries imposed by AFDX and are useful for basic interoperability testing only.

To achieve determinism, specialized hardware to enforce the *Virtual Link* rules, which are based on the VL parameters. introduced by ARINC 664 is needed. AFDX switches fill this role and enforce latency, bandwidth constraints for VLs and provide a dependable, fixed configuration. This configuration is read at bootup and remains constant at run time to avoid fluctuations in the network's topology and provide uniform timing behavior.

For integrity reasons, store-and-forward circuit switching is used when relaying packets, in contrast to most modern day high-speed Ethernet switches, which perform cut-through switching [19]. The configuration for all *Virtual Links* (LMIN, LMAX, BAG, priority) and switch parameters should be set according to a one of the mathematical proofing models in use today [26, 27].

By fixing network parameters at boot-up, changes at runtime are prevented and the network retains constant timing properties and a static layout throughout operation. Non-fault generated deviations off default settings may not happen and are taken into account when calculating global parameters mathematically [27]. Switches isolate *Virtual Links* from each other and perform scheduling for passing-through frames based on their VLID [4]. Other parameters specified in switch and system configuration include priority, LMIN (equivalent to LMAX) and jitter for *Virtual Link.* Ports have a fixed maximum delay and buffer-size [19].

Certification of components for use in avionics environments requires provable properties and usually results in a worst-case but congestion-free setup. Network Calculus [28] is widely used, but alternative approaches such as the trajectory based models or model-checking would be viable alternatives, too; common to all of them is a resulting formal proof of the network's correct functionality [26, 27].

5.2.5 Impact On OSI-Layer 3 and Above

AFDX adheres to the OSI layer model and is based on common protocols from the Internet-world. Subsequently, familiar protocols like IP, UDP and IP-multicast are used. Alien networking environments, such as ARINC 429 links, can be transported within a *Virtual Link* transparently to the individual applications, thereby reducing development effort. In fact, virtually any previous network standard which does not exceed ARINC 664 in capabilities can be implemented on top of it [19].

At Layer 3, the IPv4 protocol is deployed, though the fields usually used for source and destination IP-addresses have been reassigned, as depicted in Figure 12. The top packet-version shows an IP packet being directed to an individual system using the VLID, while the bottom packet uses multicast-addressing. The 32 bits of the source IP address field are separated into:

- the single bit *class identifier*,
- 7 bit *private address*,
- user-defined 16 bit ID,
- as well as an 8 bit *partition identifier.*

The *partition identifier* is used to address virtual *subsystems* in a virtualized IMA environment [12, 18].

The Destination IP is either used to designate a multicast

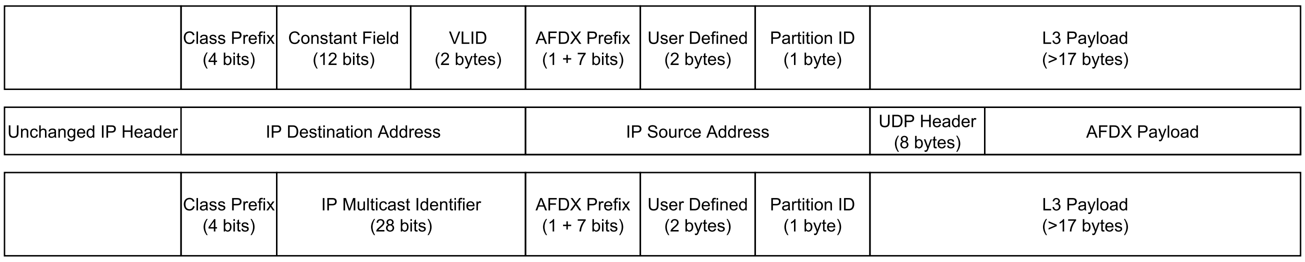


Figure 12: Similar to AFDX's frame format on OSI-layer 2, the structure of the IPv4 header remains unchanged. [19]

IP address, or contains a field of 16 bits prefixed to the VLID. The first 16 bits contain a fixed number (specified by the standard), while the second part contains the VLID, if direct IP-addressing and IMA is used [19].

Due to the guarantees provided by AFDX, certain features usually introduced at higher OSI layers (e.g. packet-loss handling and reordering of packets) are already implemented by the underlying L2/3-networking structure. In commercial networking, protocols such as TCP or SCTP are used to provide this functionality. In AFDX, transmission control and integrity is already provided at the lower layers, thus, UDP was chosen to be the default protocol in AFDX [7].

AFDX-Ports are mapped directly at UDP's source and destination port fields. AFDX-flows are identified by using a combination of the following parameters:

- destination MAC address (containing the VLID),
- source and destination IP address,
- source and destination UDP port,

Due to architectural restrictions, the minimum payload size for packets transmitted inside a AFDX-L3 packet is 144 bits. If an UDP packet's length drops below this limit, padding is added at the end of the L4 packet [19].

The standard also defines monitoring to be performed via SNMP, and intra-component data transfer through TFTP. Payload transferred inside the L4-structure usually has no fixed predetermined meaning, in contrast to earlier standards. However, ARINC 664 defines a number of common data structures, such as floating point number formats and booleans. These do have no direct impact on network payload, but offer common ground for software development [19].

6. CONCLUSIONS

ARINC 429 was developed at a time when the use of interconnected, programmable *subsystems* aboard aircraft was simply not feasible due to aspects such as size, energy consumption, fragility and hardware cost. 429 solely treats data transfer between systems at a per-device level, interconnecting systems on a pin level. Though it has advantages over more modern standards, it clearly had reached its limits once multipurpose computers are interconnected. However, 429 will most likely not simply vanish; it will still be used in scenarios where simple signaling is sufficient, and in latency critical scenarios. It is a proven and extremely reliable technology and thus is also used as fall-back network for the AFDX network, e.g. in the Airbus A380.

Most of 429's limitations have been identified decades ago, but no uniform standard had been adopted by the aerospace industry. Other standards introduced more modern, faster or more flexible networking models, in contrast to basic signaling as in 429. However, known attempts are either in use exclusively in individual aircraft models or are applied only internally by manufacturers (629, *ASCB*, *CSCB*, *EFAbus* [4]) offering little or no compatibility to the other implementations. Still, these standards introduced approaches which can be found in an altered form in AFDX [7].

AFDX combines proven safety and availability functionality with modern technology to be able to handle today's requirements. It adheres to the OSI-layer-model and outlines a compatible stack architecture, while allowing to emulate previous communication standards on top. Besides, the Internet Protocols Suite (IP/UDP) and Ethernet are used and only slight alterations to the individual data structures are applied, which lowers the bar for designing hardware and developing software in avionics considerably.

	ARINC 429	ARINC 629	Mil-Std-1553	ARINC 664 (at 100 Mbps)
Topology (logical)	Bus	Bus	Bus (Star)	Star
Duplex	Simplex	Half-Duplex	Half-Duplex	Full-Duplex
Medium	Dedicated	Shared	Shared	Shared
Speed	100 kHz	2 MHz	1 MHz	100 MHz
Bandwidth	2778 words/sec	Variable	46000 words/sec	3,000,000+ frames/sec
Latency	Fixed	Bounded	Variable	Bounded
QoS	100%	None	None	Configurable

Table 2: A capability comparison of ARINC 429, 629, Mil-Std-1553 and AFDX. [7]

For certain parts of an AFDX network, COTS hardware can be used in conjunction with matching software, though AFDX hardware implementations must be used to retain determinism. Still, by adding standard Ethernet hardware in conjunction with an AFDX-stack implementation in the operating system, non-AFDX hardware could be used without further alterations [19, 23].

Changes to the overall network layout do not negatively impact individual *Virtual Links* or ports of the individual end- and *subsystems*, due to the added abstraction [12, 18]. Diagnosing issues within an AFDX network requires highly skilled personnel, in contrast to 429. Still, hardware swapping in a 664 network can be done with little effort whereas fixing a line running through an entire aircraft due to a fault may require considerable effort [7]. ARINC 664 supporting devices may also support virtualization and hardware partitioning, as virtual/logical devices as specified in IMA/ARINC 653 can be used [29]. 429 will never be able to support IMA architectures, let alone non-physical hardware [12, 13, 18].

AFDX implementations still remain relatively conservative, using a proven and mature technology base instead of state-of-the-art hardware [14]. For example, the Airbus A380's and A350's networks are based on copper-cabling, while optical cabling has become the de-facto standard for high-speed interconnection on backbones in corporate and carrier backbones. However, ARINC 664 architectures can integrate future technology seamlessly. Future AFDX implementations like the one used in Boeing's 787 will use fiber-optics.

664's flexibility in complex setups makes it an obvious solution to the cabling overhead and complexity issues of 429. While offering a comparable level of reliability as Mil-Std-664 in civilian application, its architecture is a considerably more adaptive and can benefit seamlessly from the parallel evolution of the Ethernet standards family. Also line speeds previously unachievable in commercial avionics are now attainable and require far less development effort for both hard and software, as the technological base still remains to be Ethernet.

In the long run, the technological layering will minimize the need to review AFDX to incorporate new features, as those can be introduced via the underlay standards. AFDX is far more advanced, modern and powerful than previous standards, yet retains its flexibility. In the future, we will most likely observe an acceleration of the introduction of this new networking standard into commercial avionics.

7. REFERENCES

[1] P. Frodyma and B. Waldmann. *ARINC 429 Specification Tutorial*. AIM GmbH, 2.1 edition, 2010.

[2] Working Group. ARINC 615, P2: ARINC Airborne Computer Data Loader. Technical Report 1, Aronautical Radio, INC, June 1991.

[3] P. Frodyma, J. Furgerson, and B. Waldmann. *MIL-STD-1553 Specification Tutorial*. AIM GmbH, 2.3 edition, 2010.

[4] L. Buckwalter. *Avionics Databuses, Third Edition*. Avionics Communications Incorporated, 2008.

[5] Miguel A. Sánchez-Puebla and Jesús Carretero. A new approach for distributed computing in avionics systems. In *Proceedings of the 1st international symposium on Information and communication technologies*, ISICT 2003, pages 579 – 584. Trinity College Dublin, 2003.

[6] N. Thanthry and R. Pendse. Aviation data networks: security issues and network architecture. *Aerospace and Electronic Systems Magazine, IEEE*, 20(6):3 – 8, June 2005.

[7] T. Schuster and D. Verma. Networking concepts comparison for avionics architecture. In *Digital Avionics Systems Conference, 2008. DASC 2008. IEEE/AIAA 27th*, pages 1.D.1–1 – 1.D.1–11, October 2008.

[8] J. Craveiro, J. Rufino, C. Almeida, R. Covelo, and P. Venda. Embedded Linux in a partitioned architecture for aerospace applications. In *Computer Systems and Applications, 2009. AICCSA 2009. IEEE/ACS International Conference on*, pages 132 – 138, May 2009.

[9] M.J. Morgan. Integrated modular avionics for next-generation commercial airplanes. In *Aerospace and Electronics Conference, 1991. NAECON 1991., Proceedings of the IEEE 1991 National*, pages 43 – 49 vol.1, May 1991.

[10] C.B. Watkins and R. Walter. Transitioning from federated avionics architectures to Integrated Modular Avionics. In *Digital Avionics Systems Conference, 2007. DASC 2007. IEEE/AIAA 26th*, pages 2.A.1–1 – 2.A.1–10, October 2007.

[11] P.J. Prisaznuk. ARINC 653 role in Integrated Modular Avionics (IMA). In *Digital Avionics Systems Conference, 2008. DASC 2008. IEEE/AIAA 27th*, pages 1.E.5–1 – 1.E.5–10, October 2008.

[12] INC Aronautical Radio. *ARINC 653, P1-3: Avionics Application Software Interface*. ARINC Specification 653 Parts 1-3, November 2010.

[13] Richard L. Alena, John P. Ossenfort Iv, Kenneth I. Laws, and Andre Goforth. Communications for Integrated Modular Avionics. In *IEEE Aerospace Conference 2007*, pages 1 – 18, March 2007.

[14] R. Ramaker, W. Krug, and W. Phebus. Application of a civil Integrated Modular Architecture to military transport aircraft. In *Digital Avionics Systems Conference, 2007. DASC 2007. IEEE/AIAA 26th*, pages 2.A.4–1 – 2.A.4–10, October 2007.

[15] Michael Hegarty. A Robust Physical Layer for aircraft data networks based on MIL-STD-1553. In *SAE 2011 AeroTech Congress and Exhibition*, pages 1394 – 1401, October 2011.

[16] Working Group. 802.3-2008 IEEE Standard for Ethernet, 2008.

[17] Gitsuzo B. S. Tagawa and Marcelo Lopes de Oliveira e Souza. An overview of the Intergrated Modular Avionics (IMA) concept. In *DINCON 2011*, pages 277 – 280. Conferência Brasileira de Dinâmica, Controle e Aplicações, September 2011.

[18] John Rushby. Partitioning in avionics architectures: Requirements, mechanisms, and assurance. Final Report DOT/FAA/AR-99/58, NASA Langley Research Center and U.S. Federal Aviation Administration (US DOT), March 2000.

[19] INC Aronautical Radio. *ARINC 664, P7: Avionics Full Duplex Switched Ethernet (AFDX) Network*. ARINC Specification 664 Part 7, June 2005.

[20] M. Felser. Real-Time Ethernet - Industry Prospective. *Proceedings of the IEEE*, 93(6):1118 – 1129, June 2005.

[21] J.-D. Decotignie. Ethernet-Based Real-Time and Industrial Communications. *Proceedings of the IEEE*, 93(6):1102 – 1117, June 2005.

[22] L A N Man and Standards Committee. *802.1Q-2005 IEEE Standard for Local and metropolitan area networks: Virtual Bridged Local Area Networks*, volume 2005. IEEE, 2005.

[23] Emre Erdinc. Soft AFDX end system implementation with standard PC and Ethernet card. Master's thesis, Graduate School of Natural and Applied Sciences of the Middle East Technical University, 2010.

[24] Working Group. 802.1AX-2008 IEEE Standard for Local and Metropolitan Area Networks - Link Aggregation, 2008.

[25] Todd Lammle. *CCNA: Cisco certified network associate study guide*. Sybex, San Francisco, 2004.

[26] M. Boyer, N. Navet, and M. Fumey. Experimental assessment of timing verification techniques for AFDX. Technical report, Realtime-at-work, ONERA, Thales Avionics, 2012.

[27] Ananda Basu, Saddek Bensalem, and Marius Bozga. Verification of an AFDX Infrastructure Using Simulations and Probabilities. In *Runtime Verification*, volume 6418 of *Lecture Notes in Computer Science*, pages 330 – 344. Springer Berlin / Heidelberg, 2010.

[28] Jean-Yves Le Boudec and Patrick Thiran. *Network calculus: a theory of deterministic queuing systems for the internet*. Springer-Verlag, Berlin, Heidelberg, 2001.

[29] D. Kleidermacher and M. Wolf. MILS virtualization for Integrated Modular Avionics. In *Digital Avionics Systems Conference, 2008. DASC 2008. IEEE/AIAA 27th*, pages 1.C.3–1 – 1.C.3–8, October 2008.