LEADING THE WORLD IN AVIONICS INTERFACE SOLUTIONS

AFDX
PROTOCOL TUTORIAL

CONDOR ENGINEERING

Copyrights

AFDX / ARINC 664 Tutorial (1500-049)

Condor Engineering, Inc.
Santa Barbara, CA 93101
(805) 965-8000
(805) 963-9630 (fax)
support@condoreng.com
http://www.condoreng.com

Document Revision: May 2005
Document Version: 3.0

CONDOR ENGINEERING

Contents and Tables

Contents

Figures

Tables

CONDOR ENGINEERING

CHAPTER 1

Overview

The Antecedents

Moving information between avionics subsystems on board an aircraft has never been more crucial, and it is here that electronic data transfer is playing a greater role than ever before. Since its entry into commercial airplane service on the Airbus A320 in 1988, the all-electronic fly-by-wire system has gained such popularity that it is becoming the only control system used on new airliners.

But there are a host of other systems — inertial platforms, communication systems, and the like — on aircraft, that demand high-reliability, high-speed communications, as well. Control systems and avionics in particular, rely on having complete and up-to-date data delivered from source to receiver in a timely fashion. For safety-critical systems, reliable real-time communications links are essential.

That is where AFDX comes in. Initiated by Airbus in the evolution of its A380 Aircraft, they coined the term, AFDX, for Avionics Full-DupleX, switched Ethernet. AFDX brings a number of improvements such as higher-speed data transfer — and with regard to the host airframe — significantly less wiring, thereby reducing wire runs and the attendant weight.

What is AFDX?

Avionics **F**ull **D**uple**X** Switched Ethernet (AFDX) is a standard that defines the electrical and protocol specifications (IEEE 802.3 and ARINC 664, Part 7) for the exchange of data between Avionics Subsystems. One thousand times faster than its predecessor, ARINC 429, it builds upon the original AFDX concepts introduced by Airbus.

One of the reasons that AFDX is such an attractive technology is that it is based upon Ethernet, a mature technology that has been continually enhanced, ever since its inception in 1972. In fact, the commercial investment and advancements in Ethernet have been huge compared say, to ARINC 429, MIL-STD-1553, and other specialized data-communications technologies.

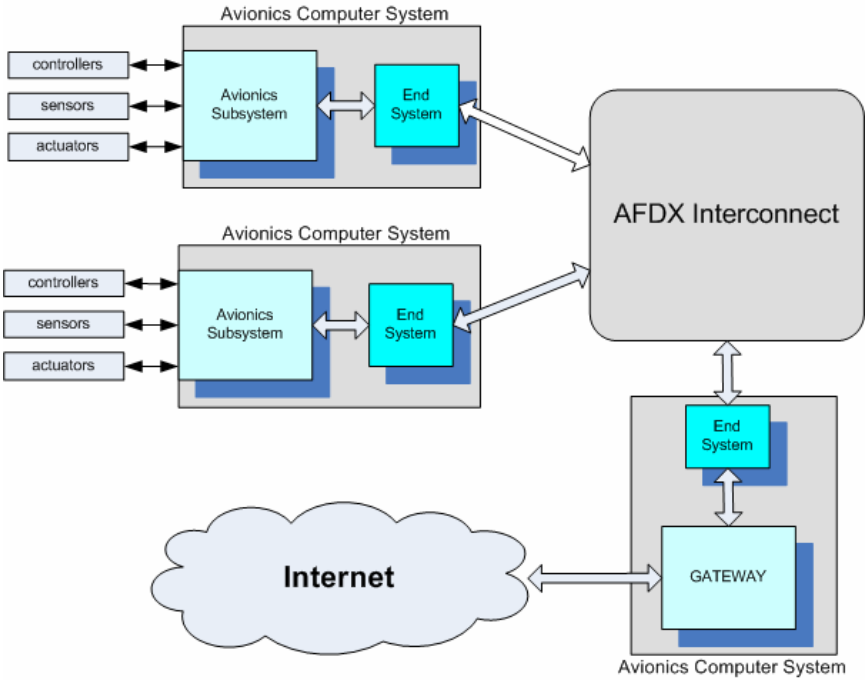

Figure 1. AFDX Network

As shown in Figure 1, an AFDX system comprises the following components:

- **Avionics Subsystem:** The traditional Avionics Subsystems on board an aircraft, such as the flight control computer, global positioning system, tire pressure monitoring system, etc. An *Avionics Computer System* provides a computational environment for the Avionics Subsystems. Each Avionics Computer System contains an embedded End System that connects the Avionics Subsystems to an AFDX Interconnect.
- **AFDX End System (End System):** Provides an "interface" between the Avionics Subsystems and the AFDX Interconnect. Each Avionics Subsystem the End System interface to guarantee a secure and reliable data interchange with other Avionics Subsystems. This interface exports an application program interface *(*API) to the various Avionics Subsystems, enabling them to communicate with each other through a simple message interface.
- **AFDX Interconnect:** A full-duplex, switched Ethernet interconnect. It generally consists of a network of switches that forward Ethernet frames to their appropriate destinations. This switched Ethernet technology is a departure from the traditional ARINC 429 unidirectional, point-to-point technology and the MIL-STD-1553 bus technology.

As shown in the example in Figure 1, two of the End Systems provide communication interfaces for three avionics subsystems and the third End System supplies an interface for a Gateway application. It, in turn, provides a communications path between the Avionics Subsystems and the external IP network and, typically, is used for data loading and logging.

The following sections provide an overview of the AFDX architecture and protocol. But first we briefly review two of the traditional avionics communications protocols.

Other Avionics Buses

This section compares AFDX to two earlier Avionics data communication protocols: ARINC 429 and MIL-STD-1553.

ARINC 429

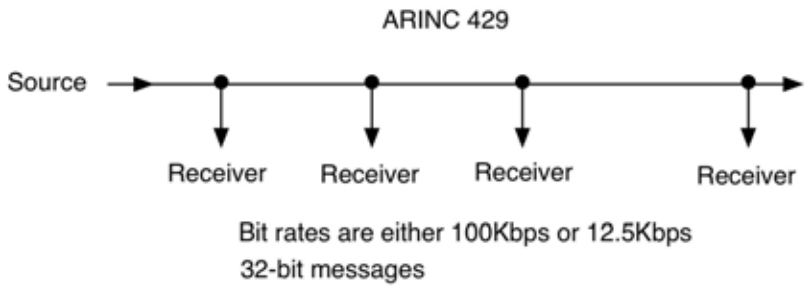

Figure 2. ARINC 429 Communication Protocol

ARINC 429 implements a single-source, multi-drop bus with up to 20 receivers (see Figure 2). Messages consist of 32-bit words with a format that includes five primary fields. The Label field determines the interpretation of the fields in the remainder of the word, including the method of translation. The point to multi-point property of ARINC 429 requires the Avionics system to include an ARINC 429 bus for each pair-wise communication. Refer to the Condor Engineering ARINC Tutorial for more details.

MIL-STD-1553

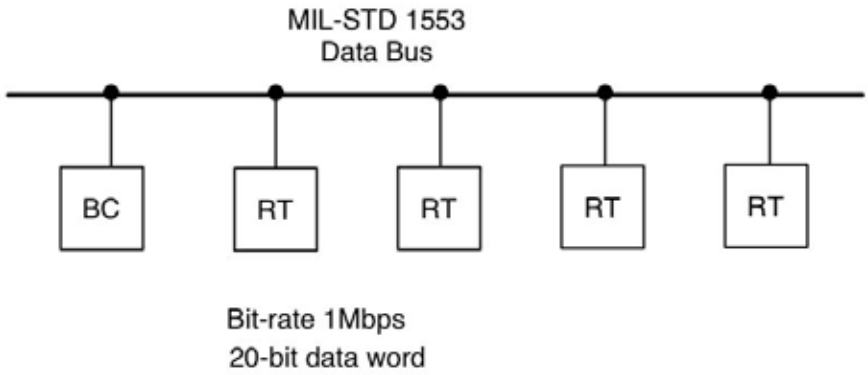

Figure 3. MIL-STD-1553 Bus Communication Protocol

MIL-STD-1553 (see Figure 3) implements a bus architecture in which all the devices attached to the bus are capable of receiving and transmitting data. The Avionics subsystems attach to the bus through an interface called a remote terminal (RT). The Tx and Rx activity of the bus is managed by a bus controller, that acts to ensure that no two devices ever transmit simultaneously on the bus. The communication is half duplex and asynchronous. For more information, refer to the Condor Engineering “MIL-STD-1553 Tutorial”.

CHAPTER 2

Ethernet

Ethernet

This chapter provides a brief description of the origins of Ethernet, the Ethernet frame format and the role of switched Ethernet in avionics applications.

ALOHA Net

In 1970, the University of Hawaii deployed a packet radio system called the "ALOHA network" [Norman Abramson; see Figure 4] to provide data communications between stations located on different islands. There was no centralized control among the stations; thus, the potential for *collisions* (simultaneous transmission by two or more stations) existed.

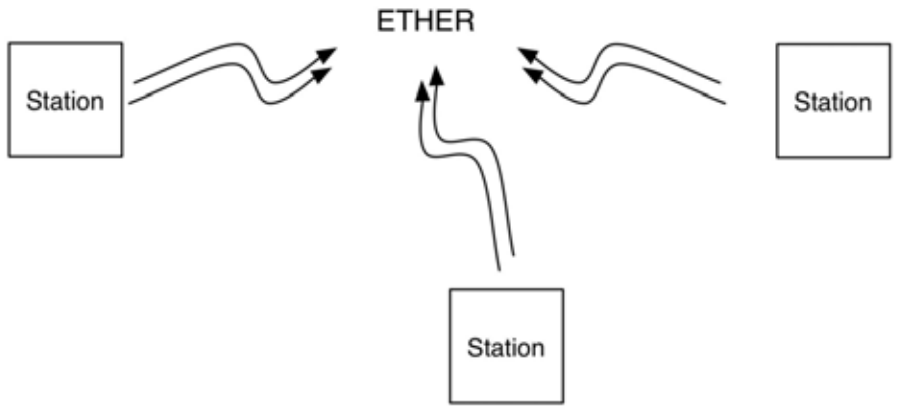

Figure 4. ALOHA Net

The ALOHA Protocol

1. If you have a message to send, send the message, and
2. If the message collides with another transmission, try resending the message later using a back-off strategy.

Issues

- No central coordination.
- Collisions lead to non-deterministic behavior.

Ethernet Local Area Networks (Broadcast Media)

In 1972, Robert Metcalfe and David Boggs at Xerox Palo Alto Research Center built upon the ALOHA network idea and used a coaxial cable as the communication medium and invented Ethernet (see Figure 5). Ethernet is similar to the ALOHA protocol in the sense that there is no centralized control and transmissions from different *stations* (*hosts*) could collide.

The Ethernet communication protocol is referred to as "CSMA / CD" (Carrier Sense, Multiple Access, and Collision Detection). *Carrier Sense* means that the hosts can detect whether the medium (coaxial cable) is idle or busy. *Multiple Access* means that multiple hosts can be connected to the common medium. *Collision Detection* means that, when a host transmits, it can detect whether its transmission has collided with the transmission of another host (or hosts). The original Ethernet data rate was 2.94Mbps.

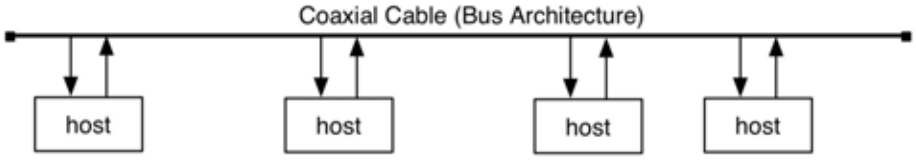

Figure 5. Ethernet Local Area Networks (Broadcast Media)

The Ethernet Protocol

1. If you have a message to send and the medium is idle, send the message.
2. If the message collides with another transmission, try sending the message later using a suitable back-off strategy.

Issues

- No central coordination.
- Collisions lead to non-deterministic behavior.

Ethernet Using Category 5 UTP Copper Twisted Pairs

The most common electrical form of Ethernet today is based on the use of twisted pair copper cables. Typically, cables are point-to-point, with hosts directly connected to a switch. In the case of Fast Ethernet (100Mbps), two pairs of Category 5 UTP copper wire are used for Tx and Rx, respectively. In the case of transmission, each 4-bit *nibble* of data is encoded by 5 bits prior to transmission. This is referred to as "4B/5B encoding" and results in a transmission clock frequency of 125Mbps, since 5 bits are sent for every 4 bits of data. Since there are twice as many 5-bit patterns as 4-bit ones, it is possible to ensure that every transmitted pattern is able to provide good clock synchronization (not too many 0’s or 1’s in a row) for reliable transmission of data. Some of the 5-bit patterns are used to represent control codes.

Ethernet Frame Format

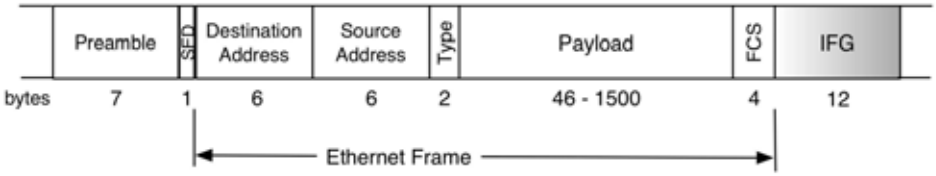

Figure 6. Ethernet Frame Format

As Figure 6 illustrates, IEEE 802.3 defines the format of an Ethernet transmission to include a 7-byte Preamble, a Start Frame Delimiter (SFD), the Ethernet frame itself, and an Inter-Frame Gap (IFG) consisting of at least 12 bytes of idle symbols. The Ethernet frame begins with the Ethernet header, which consists of a 6-byte destination address, followed by a 6-byte source address, and a type field. The Ethernet payload follows the header. The frame concludes with a Frame Check Sequence (FCS) for detecting bit errors in the transmitted frame, followed by an IFG. The length of an Ethernet frame can vary from a minimum of 64 bytes to a maximum of 1518 bytes.

Ethernet communication (at the link level) is connectionless. Acknowledgments must be handled at higher levels in the protocol stack.

Full-duplex, Switched Ethernet

The Scenario

Half-duplex Mode Ethernet is another name for the original Ethernet Local Area Network discussed earlier. As we explained, there is an issue when multiple hosts are connected to the same communication medium as is the case with coaxial cable, depicted in Figure 5, and there is no central coordination. It is possible for two hosts to transmit "simultaneously" so that their transmissions "collide." Thus there is a need for the hosts to be able to detect transmission collisions. When a collision occurs (two or more hosts attempting to transmit at the same time), each host has to retransmit its data. Clearly, there is a possibility that they will retransmit at the same time, and their transmissions will again collide.

To avoid this phenomenon, each host selects a random transmission time from an interval for retransmitting the data. If a collision is again detected, the hosts selects another random time for transmission from an interval that is twice the size of the previous one, and so on. This is often referred to as the *binary exponential backoff strategy*.

Since there is no central control in Ethernet and in spite of the random elements in the binary exponential backoff strategy, it is theoretically possible for the packets to repeatedly collide. What this means is that in trying to transmit a single packet, there is a chance that you could have an infinite chain of collisions, and the packet would never be successfully transmitted.

Therefore, in half-duplex mode it is possible for there to be very large transmission delays due to collisions. This situation is unacceptable in an avionics data network.

So, what was required (and what was implemented in AFDX) was an architecture in which the maximum amount of time it would take any one packet to reach its destination is known. That meant ridding the system of contention.

Doing Away with Contention

To do away with contention (collisions), and hence the indeterminacy regarding how long a packet takes to travel from sender to receiver, it is necessary to move to Full-duplex Switched Ethernet. Full-duplex Switched Ethernet eliminates the possibility of transmission collisions like the ones that occur when using Half-duplex Based Ethernet. As shown in Figure 7, each Avionics Subsystem— autopilot, heads-up display, etc. — is directly connected to a Switch over a full-duplex link that comprises two twisted pairs — one pair for transmit (Tx) and one pair for receive (Rx). (The switch comprises all the components contained in the large box.) The switch is able to buffer packets for both reception and transmission.

Now, let's look at what goes on within the switch, as shown in Figure 7.

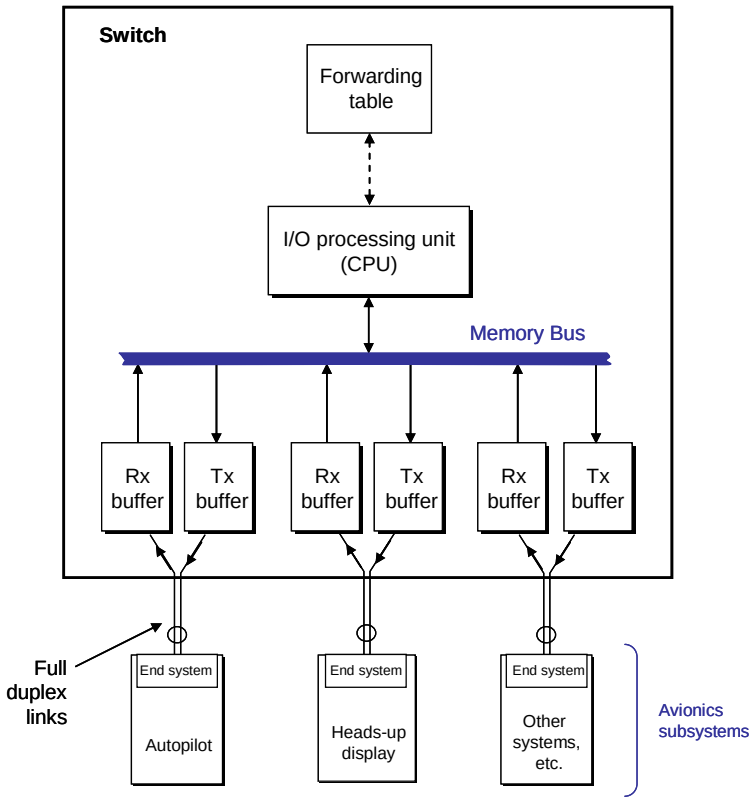

Figure 7. Full-Duplex, Switched Ethernet Example

The Rx and Tx buffers in the switch are both capable of storing multiple incoming/outgoing packets in FIFO (first-in, first out) order. The role of the I/O processing unit (CPU) is to move packets from the incoming Rx buffers to the outgoing Tx buffers. It does this by examining each arriving packet that is next in line in the Rx buffer to determine its destination address (virtual link identifier) and then goes to the Forwarding Table to determine which Tx buffers are to receive the packet. The packet is then copied into the Tx buffers, through the Memory Bus, and transmitted (in FIFO order) on the outgoing link to the selected Avionic Subsystem or to another switch. This type of switching architecture is referred to as *store and forward*.

Consequently, with this full-duplex switch architecture the contention encountered with half-duplex Ethernet is eliminated, simply because the architecture eliminates collisions.

Theoretically, a Rx or Tx buffer could overflow, but if the buffer requirement in an avionics system are sized correctly, overflow can be avoided.

There are no collisions with full-duplex switched Ethernet, but packets may experience delay due to congestion in the switch.

Instead of collisions and retransmissions, switching architecture may result in jitter, due to the random delay introduced by one packet waiting for

another to be transmitted. The extent of jitter introduced by an End System and Switch must be controlled if *deterministic* behavior of the overall Avionics System is to be achieved.

Reducing Wire Runs and Weight

In addition to the enhancements already described, AFDX delivers some additional benefits, compared to ARINC 429. Figure 8 shows some distinctions between ARINC 429 and AFDX. In ARINC 429, a twisted pair must link every device that receives the azimuth signal form the inertial platform. The point-to-multi-point and unidirectional properties of ARINC 429 means that the avionics system must include an ARINC 429 bus for each communication path. In a system with many end points, point-to-point wiring is a major overhead. This can lead to some huge wiring harnesses, with the added weight that goes along with them.

But in the case of AFDX, as shown in Figure 8b, each signal is connected to the switch only once so that no matter how many subsystems require the azimuth signal from the inertial platform, they need not be connected individually to the inertial platform.

With ARINC 429, a transmitter can fan out to only 20 receivers. With AFDX, the number of fan-outs from the inertial platform is limited only by the number of ports on the switch. Also, by cascading switches, the fan-out can be easily increased as needed.

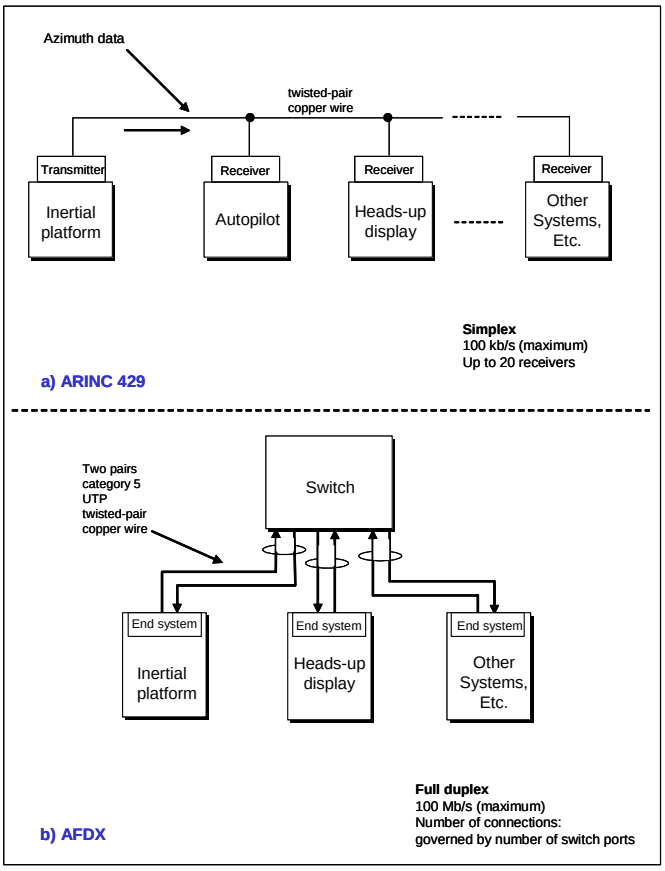

Figure 8. AFDX versus ARINC 429 architecture

CHAPTER 3

End Systems and Avionics Subsystems

End Systems and Avionics Subsystems

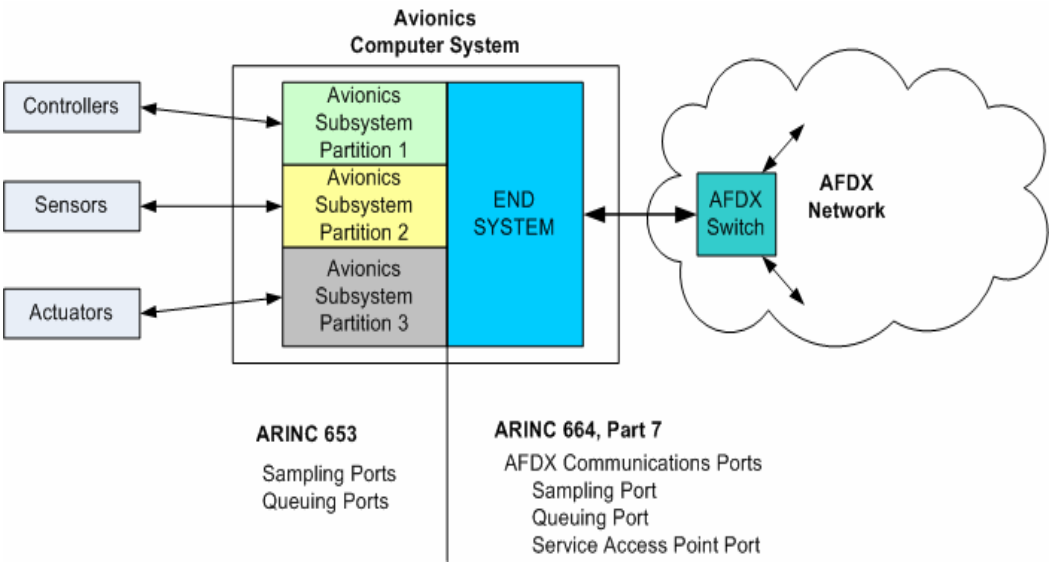

Figure 9. End Systems and Avionics Subsystems Example

As Figure 9 shows, an Avionics computer system connects to the AFDX network through an End System. In general, an Avionics computer system is capable of supporting multiple Avionics subsystems. Partitions provide isolation between Avionics subsystems within the same Avionics computer system. This isolation is achieved by restricting the address space of each partition and by placing limits on the amount of CPU time allotted to each partition. The objective is to ensure that an errant Avionics subsystem running in one partition will not affect subsystems running in other partitions.

End Systems and Avionics Subsystems

Avionics applications communicate with each other by sending messages using communication ports. The specification of an operating system API for writing portable avionics applications can be found in ARINC 653. In particular, ARINC 653 defines two types of communications ports – *sampling* and *queuing* ports. Accordingly, it is necessary that End Systems provide a suitable communications interface for supporting sampling and queuing ports. The AFDX ports, defined in ARINC 664, Part 7, include sampling, queuing and SAP ports. The AFDX sampling and queuing ports correspond to ARINC 653 sampling and queuing ports, respectively. AFDX introduces a third port type called a Service Access Point (SAP) port. SAP ports are used for communications between AFDX system components and non-AFDX systems. More about this in the next chapter.

End Systems are identified using two 8-bit quantities: a Network ID and an Equipment ID. These may be combined into a single 16-bit quantity. As we shall see, the End System identification is used in forming source MAC addresses and unicast IP addresses.

CHAPTER 4

AFDX Communications Ports

AFDX Communications Ports

Avionics subsystems use communications ports to send messages to each other. Communication ports, which are typically part of the operating system API, provide a programming mechanism for sending and receiving messages. Two types of communications ports play a role in Avionics subsystems: sampling and queuing ports.

AFDX End Systems must provide both sampling and queuing port services, as described in ARINC 653.

As Figure 10 and Figure 11 show, sampling and queuing ports differ mainly in reception. A sampling port has buffer storage for a single message; arriving messages overwrite the message currently stored in the buffer. Reading a message from a sampling port does not remove the message from the buffer, and therefore it can be read repeatedly. Each sampling port must provide an indication of the *freshness* of the message contained in the port buffer. Without this indication, it would be impossible to tell whether the transmitting Avionics subsystem has stopped transmitting or is repeatedly sending the same message.

A queuing port has sufficient storage for a fixed number of messages (a configuration parameter), and new messages are appended to the queue. Reading from a queuing port removes the message from the queue (FIFO).

Typical programming interfaces for sending and receiving messages are as follows:

- Send_Msg(port_ID, message)
- Recv_Msg(port_ID, message)

The port_ID identifies the communication port, and the message argument points to a buffer that either contains the message to be sent or is available to receive a new message from the port.

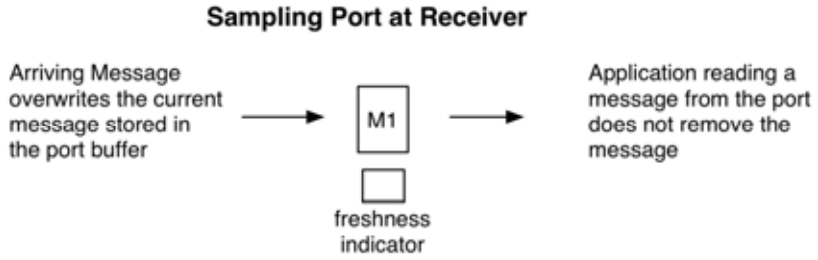

Figure 10. Sampling Port at Receiver

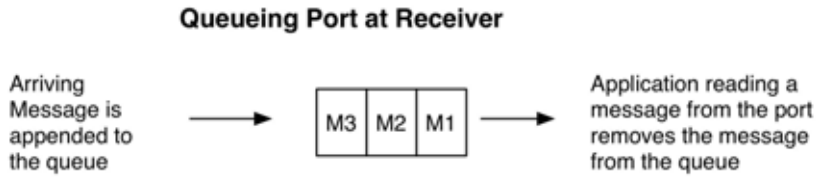

Figure 11. Queuing Port at Receiver

CHAPTER 5

Virtual Links: Packet Routing in AFDX

Virtual Links

In a traditional Ethernet switch, incoming Ethernet frames are routed to output links based on the Ethernet destination address. In AFDX, a 16-bit value called a Virtual Link ID is used to route Ethernet frames in an AFDX network. Figure 12 provides the format of the Ethernet destination address in an AFDX network.

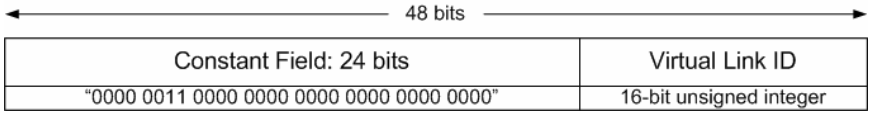

Figure 12. Format of Ethernet Destination Address in AFDX Network

The switches in an AFDX network are "configured" to route an incoming Ethernet frame to one or more outgoing links. An important property of an AFDX network is that Ethernet frames associated with a particular Virtual Link ID *must* originate at one, and only one, End System. The AFDX switches are configured to deliver frames with the same Virtual Link ID to a predetermined set of End Systems. Thus, a virtual link originates at a single End System and delivers packets to a fixed set of End Systems; this is analogous to an ARINC 429 multi-drop bus.

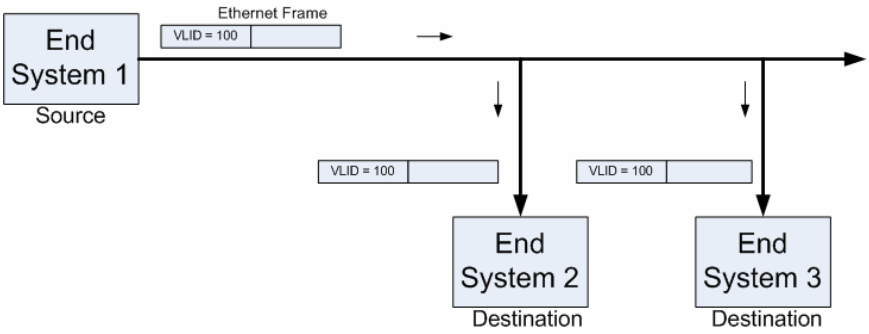

Figure 13. Packet Routing Example

In the example in Figure 13, when the Source End System (1) sends an Ethernet frame with a Virtual Link ID (VLID) = 100 to the network, the AFDX switches deliver the frame to a predetermined set of destination End Systems (2 and 3). More than one virtual link can originate at an End System, and each virtual link can carry messages from one or more communication ports.

CHAPTER 6

Message Flows

Message Flows

When an application sends a message to a communications port, the source End System, the AFDX network, and the destination End Systems are configured to deliver the message to the appropriate receive ports.

Figure 14 shows a message M being sent to Port 1 by the Avionics subsystem. End System 1 encapsulates the message in an Ethernet frame and sends the Ethernet frame to the AFDX Switched Network on Virtual Link 100 (the Ethernet destination address specifies VLID 100). The forwarding tables in the network switches are configured to deliver the Ethernet frame to both End System 2 and End System 3. The End Systems that receive the Ethernet frame are configured so that they are able to determine the destination ports for the message contained in the Ethernet frame. In the case shown in Figure 14 the message is delivered by End System 2 to port 5 and by End System 3 to port 6.

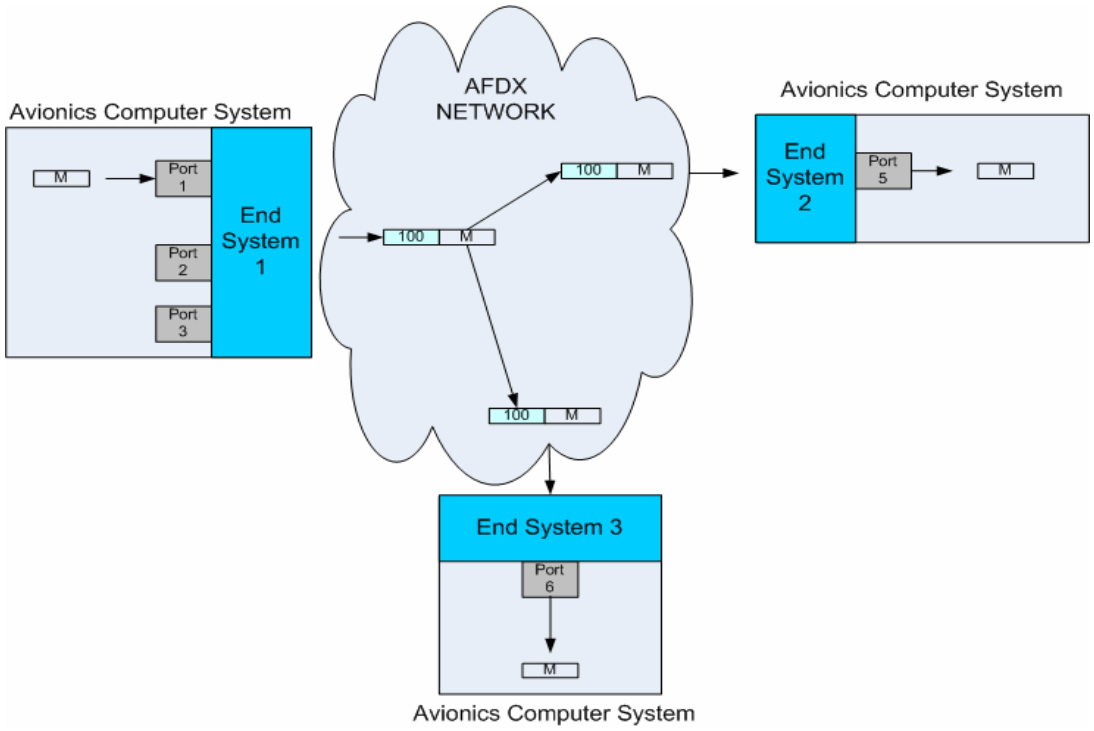

Figure 14. Message Sent to Port 1 by the Avionics Subsystem

The information used by the destination End System to find the appropriate destination port for the message is contained in headers within the Ethernet payload.

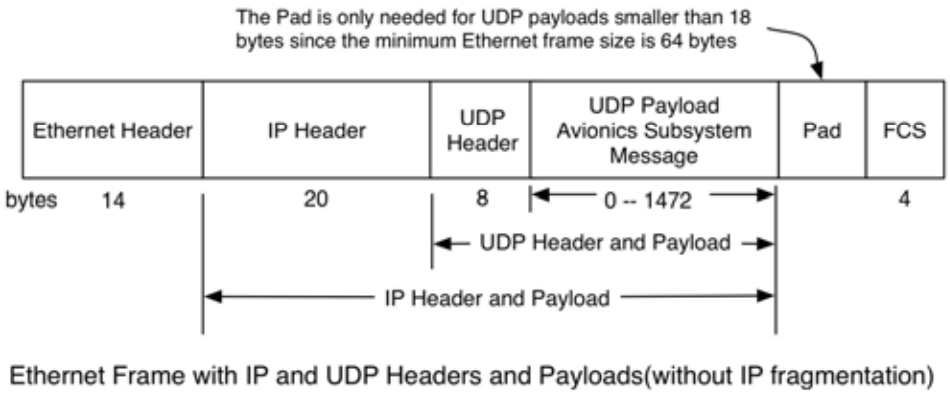

Figure 15. Ethernet Frame with IP and UDP Headers and Payloads

Figure 15 shows the headers that make up the Ethernet payload. The Ethernet payload consists of the IP packet (header and payload). The IP packet payload contains the UDP packet (header and payload), which contains the message sent by the Avionics subsystem. The Pad is necessary only when the UDP payload is smaller than 18 bytes; in this case, the pad plus the UDP payload will equal 18 bytes. With UDP payloads greater than or equal to 18 bytes, there is no Pad field. Note that the diagram applies only to UDP payloads that have not been fragmented among multiple IP payloads. An important function of the IP header is to provide fragmentation control for *large* UDP packets.

The IP header contains a destination End System Identification and partition identifiers or is a multicast address. In the latter case, the IP destination address contains the Virtual Link ID (the Virtual Link ID in the Destination Ethernet address). The UDP header contains both source and destination UDP port numbers. In general, there is enough information in these headers for an End System to determine the destination port for the message. Similarly, sufficient information associated with a transmitting AFDX communication port exists for the source End System to construct the appropriate headers when building the Ethernet frame that contains the message.

Appendix A, "AFDX Frame Addressing and Header Structures", provides additional details on the contents and format of the Ethernet frames.

CHAPTER 7

Redundancy Management

Redundancy Management

There are two independent switched networks in an AFDX system, the A and B Networks. Each packet transmitted by an End System is sent on both networks. Therefore, under normal operation, each End System will receive two copies of each packet (see Figure 16).

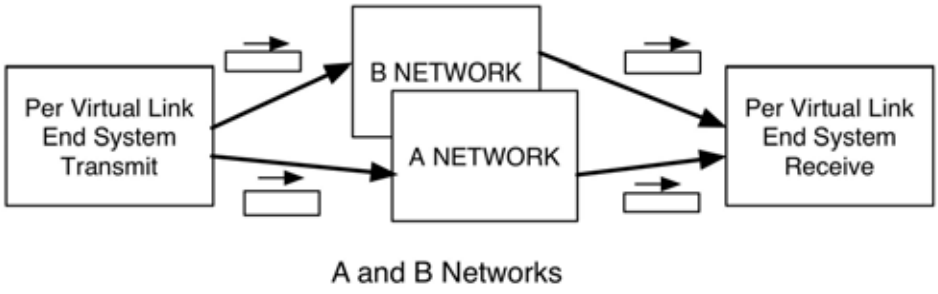

Figure 16. A and B Networks

End Systems need a way to identify corresponding packets (replicas) that arrive on the A and B networks. In AFDX, all packets transmitted over virtual link are provided with a 1-byte sequence number field. The sequence number field appears just before the FCS field in the Ethernet frame. This means that, in AFDX, one less byte is available to the IP/UDP payload. The sequence numbers start with 0, continue to 255, and then roll over to 1. The sequence number 0 is reserved for *End System Reset*. Sequence numbers are provided on a per-Virtual Link basis. An Ethernet frame with an embedded sequence number is referred to as an AFDX frame.

Redundancy Management

Figure 17 applies when the UDP payload is between 17 and 1471 bytes. If the UDP payload is smaller than 17 bytes, then a Pad field is introduced between the UDP Payload and the Sequence Number fields.

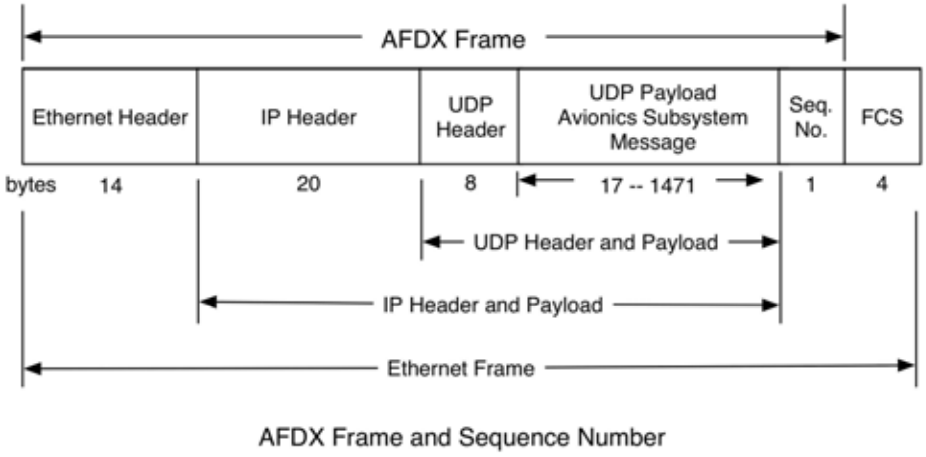

Figure 17. AFDX Frame and Sequence Number

On a per-virtual link and per-network port basis, the receiving End System checks that the sequence numbers on successive frames are in order. This is referred to as "Integrity Checking." After Integrity Checking is complete, the End System determines whether to pass the packet along or drop it because its replica has already been sent along. This is called *Redundancy Management*. Figure 18 summarizes the process.

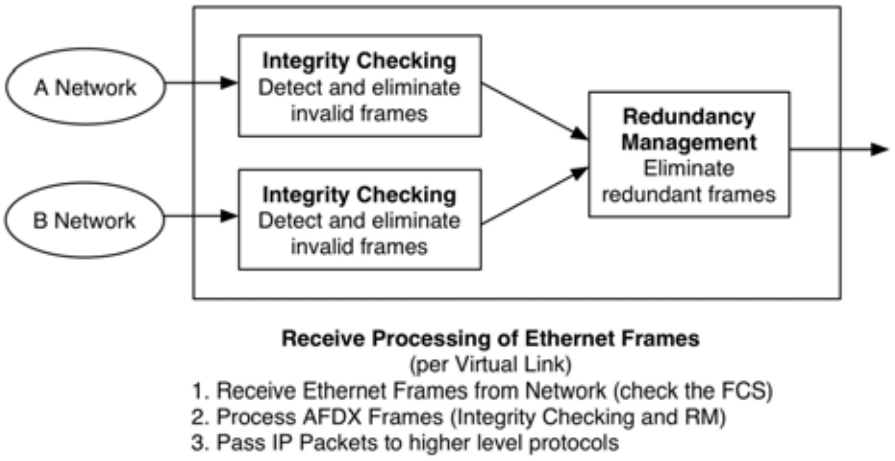

Figure 18. Receive Processing of Ethernet Frames

CHAPTER 8

Virtual Link Isolation

Virtual Link Isolation

As mentioned previously, the 100 Mbps link of an End System can support multiple virtual links. These virtual links share the 100 Mbps bandwidth of the physical link. Figure 19 shows three virtual links being carried by a single 100Mbps physical link. The figure also shows that the messages sent on AFDX Ports 1, 2, and 3 are carried by Virtual Link 1. Messages sent on AFDX Ports 6 and 7 are carried by Virtual Link 2, and messages sent on AFDX Ports 4 and 5 are carried by Virtual Link 3.

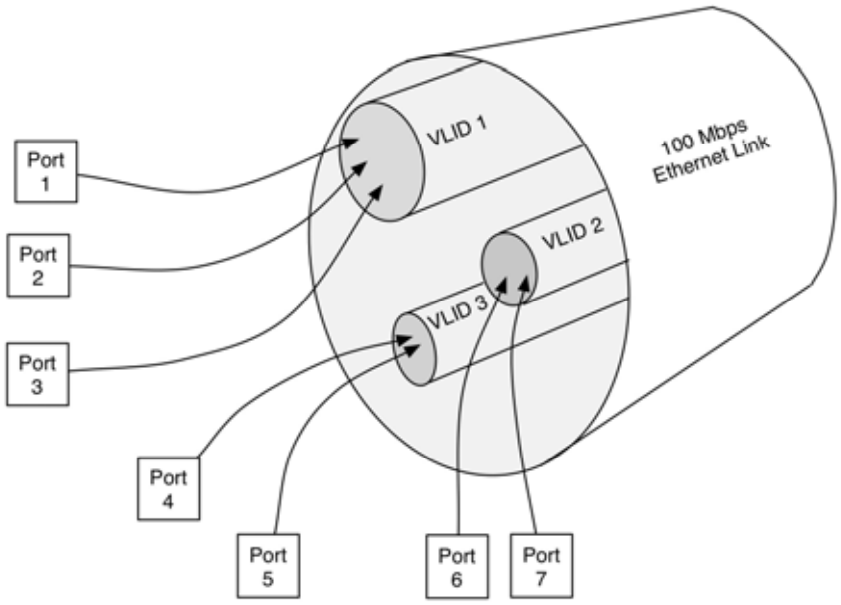

Figure 19. Three Virtual Links Carried by a Physical Link

Virtual Link Isolation

Just as partitions are used to isolate Avionics subsystems from one another, a similar mechanism is required to isolate individual virtual links, in order to prevent the traffic on one virtual link from interfering with traffic on other virtual links using the same physical link. This is done by limiting the rate at which Ethernet frames can be transmitted on a virtual link and by limiting the size of the Ethernet frames that can be transmitted on a virtual link.

Each virtual link is assigned two parameters:

- Bandwidth Allocation Gap (BAG), a value ranging in powers of 2 from 1 to 128 milliseconds
- Lmax, the largest Ethernet frame, in bytes, that can be transmitted on the virtual link

The BAG represents the minimum interval in milliseconds between Ethernet frames that are transmitted on the virtual link. For example, if a virtual link with VLID 1 has a BAG of 32 milliseconds, then Ethernet packets are never sent faster than one packet every 32 milliseconds on VLID 1. If VLID 1 has an Lmax of 200 bytes, then the maximum bandwidth on VLID 1 is 50,000 bits per second (200* 8*1000/32).

The table below indicates the allowable values for the BAG and the corresponding frequencies.

Table 1. Allowable BAG Values

BAG milliseconds	Hz
1	**1000**
2	**500**
4	**250**
8	**125**
16	**62.5**
32	**31.25**
64	**15.625**
128	**7.8125**

Choosing the BAG and Lmax for a Virtual Link

The choice of BAG for a particular virtual link depends on the requirements of the AFDX ports that are being provided link-level transport by the virtual link. For example, suppose an Avionics subsystem is sending messages on three AFDX communications ports that are being carried by the same virtual link. Let's assume the message frequencies on the ports are 10 Hz, 20 Hz, and 40 Hz, respectively. The total frequency of the combined messages (they will be combined on the same virtual link) is 70 Hz. The average period of the message transmissions is 14.4ms. Accordingly, to provide adequate bandwidth on the virtual link, we should select a BAG that is less than 14.4 ms. The first available BAG is 8 ms, which corresponds to a frequency of 125 Hz. With a BAG of 8 ms, we are guaranteed that the virtual link is able to transport the combined messages from the three ports without any backlog.

The source End System is required to enforce BAG restrictions for each outgoing virtual link. A number of virtual link scheduling algorithms can be used by the End System.

Lmax should be chosen to accommodate the largest Ethernet frame to be transmitted by these ports on the virtual link.

CHAPTER 9

Virtual Link Scheduling

Virtual Link Scheduling

Each sending AFDX communication port is associated with a virtual link. Messages sent to the communication port are encapsulated within UDP, IP, and Ethernet headers and placed in the appropriate virtual link queue for transmission. The transmission of Ethernet frames in a virtual link queue is scheduled by the End System's Virtual Link Scheduler. The Virtual Link Scheduler is responsible for scheduling transmissions of all the virtual links originating with this End System.

Figure 20 summarizes the Virtual Link Scheduling scenario. The Virtual Link Scheduler is responsible for ensuring that each virtual link conforms to its assigned bandwidth limitation. Not only must the Virtual Link Scheduler ensure the BAG and Lmax limitations for each virtual link, but it is also responsible for multiplexing all of the virtual link transmissions so that the amount of jitter introduced by the multiplexing is within acceptable bounds.

Virtual Link Scheduling

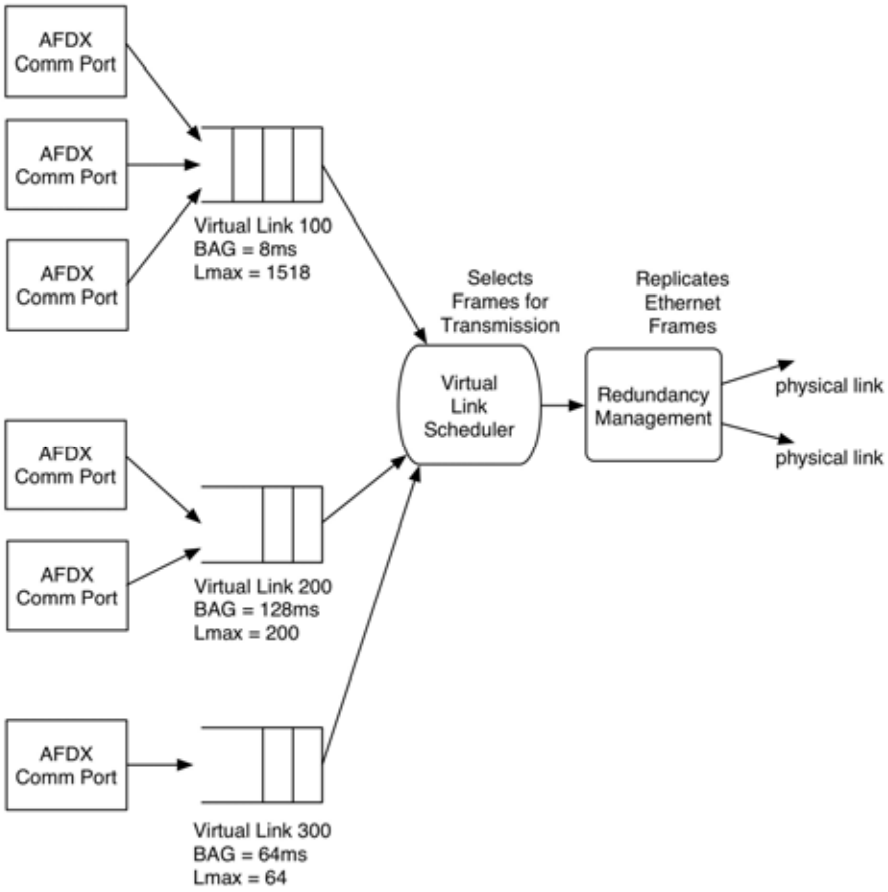

Figure 20. Virtual Link Scheduling

The timing of messages sent to an AFDX communications port is controlled by the Avionics subsystems and requirements of various attached devices. For example, a sensor may be sending readings at a 10 Hz rate. Jitter may be introduced if the message arrives at a non-empty virtual link queue. Similarly, multiplexing all the virtual link queues into the Redundancy Management Unit and subsequent transmission on the physical links can introduce additional jitter.

The ARINC 664 specification requires that, in transmission, the maximum allowed jitter on each virtual link at the output of the End System comply with both of the following formulas:

$$\max\_ jitter \le 40\mu s + \frac{\sum_{j \in \{set\ of\ VLs\}} \left(20 + L\max_j\right) \times 8)}{Nbw}$$

$$max\_ jitter \le 500\mu s$$

Nbw is the link bandwidth (100 Mbps). The first formula represents a bound on the jitter arising from an Ethernet frame being delayed by a

frame from each of the other virtual links. The second formula is a hard limit, independent of the number of virtual links. These requirements are necessary to demonstrate the overall "determinism" of an AFDX network.

Once a frame is selected from a virtual link queue for transmission, the per-VL sequence number is assigned and the frame is sent to the Redundancy Management Unit for replication (if necessary) and transmission on the physical link(s). The sequence numbers are not assigned to AFDX frames sooner than actual virtual link scheduling because of the sub-VL mechanism. If a virtual link has more than one sub-VL, the sequence number cannot be assigned to an AFDX frame until it is actually chosen by the Virtual Link Scheduler for transmission.

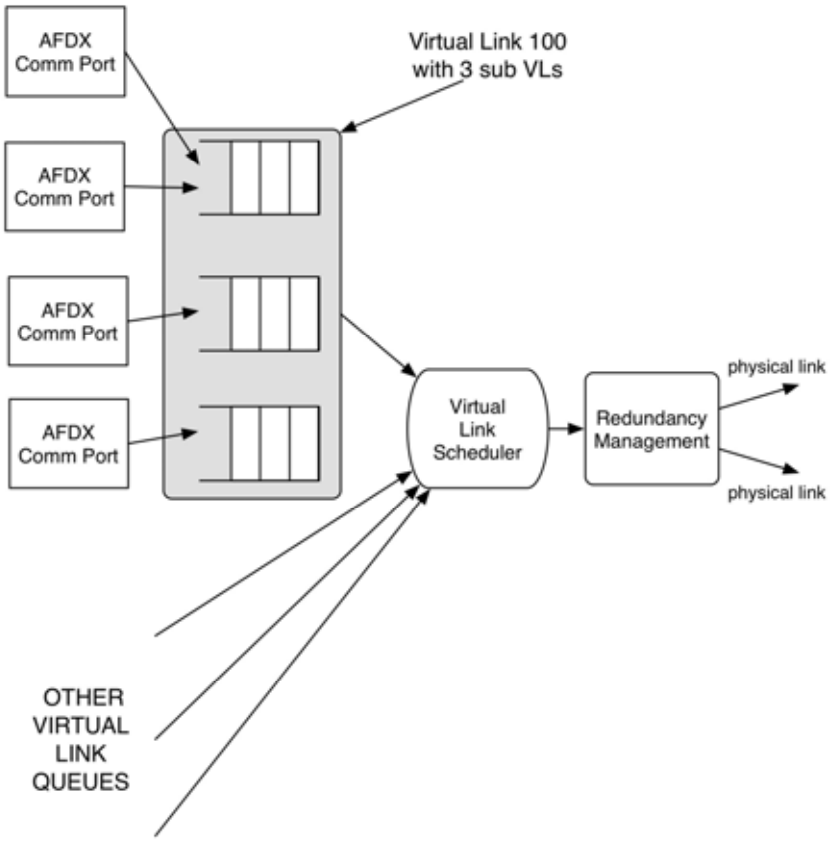

Figure 21. Virtual Link Scheduling

Figure 21 depicts a virtual link with three sub-VLs. The Virtual Link Scheduler treats these sub-VLs as a single virtual link for scheduling purposes. However, packets are selected for transmission from the sub-VL queues in a round-robin manner. Clearly, sequence numbers cannot be assigned to the packets in the sub-VL queues until they are actually selected for transmission by the Virtual Link Scheduler. If there is only a single sub-VL (the usual case), sequence numbers can be assigned as the Ethernet frames are inserted in the virtual link queue.

CHAPTER 10

Jitter

Jitter

Virtual Link scheduling consists of two components: packet regulation and multiplexing.

Figure 22 shows the role of the Virtual Link Regulators in *pacing* the frames from the virtual link queues to create zero-jitter output streams. The Virtual Link Scheduler is also responsible for *multiplexing* the regulator outputs into the Redundancy Management Unit for replication and transmission on the physical links.

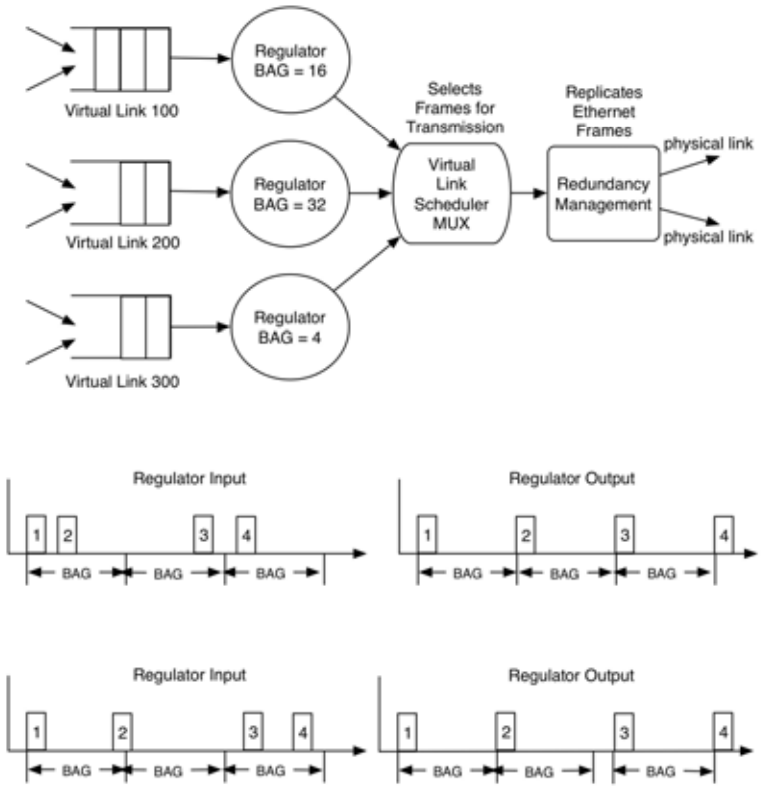

Figure 22. Role of Virtual Link Regulation

The outputs of the Regulator consist of *regulated* streams of Ethernet frames. Jitter is introduced when the Regulator outputs are combined by the Virtual Link Scheduler MUX; Ethernet frames arriving at input to the MUX at the same time will experience queuing delay (jitter).

CHAPTER 11

AFDX Message Structures

Introduction

Avionics subsystem designers are reasonably free to choose the message structure that bests suits the Avionics application. The messages are contained in the payload of the UDP packet. In general, the interpretation of the messages is determined by agreement between the Avionics applications.

ARINC 664, Part 7, identifies two types of message structures: explicit and implicit. Explicit message structures include format information that enables the receiver to correctly interpret the data. Implicit message structures do not contain any descriptive information to aid the receiver in interpreting the data; consequently, they use network bandwidth more efficiently.

This section discusses the ARINC 664 Implicit Message Structure formats. Since there is no explicit format information contained in an implicit message structure, the Avionics application needs a way to identify the message format of the received data. This is accomplished by associating implicit message structures with an AFDX receive port. The application associates the message structure based on the UDP port number where the message is received.

With the Internet, certain well-known UDP port numbers correspond to specific applications: port 69 is used by the Trivial File Transport Protocol (TFTP); port 80 is used by the Hypertext Transport Protocol (HTTP), and so on. An Internet Assigned Number Authority (IANA) manages the space of UPD port numbers. UPD port numbers fall into three groups:

- Assigned port numbers (well-known ports): 0–1023
- Registered port numbers: 1024–4951
- Dynamic/Private port numbers: 49152–65535

Although AFDX/ARINC 664 is a *closed* network, UDP port numbers should be selected from the Dynamic/Private range of numbers. The reason for this is that there could be potential conflicts with the standard port number assignments when a *gateway* is used to communicate between the AFDX network and the Internet.

Implicit Message Structures

ARINC 664, Part 7, presents a more complete description of the format of implicit message structures. A limited number of data types are defined, including the following:

- Signed_32 Integer
- Signed_64 Integer
- Float_32
- Float_64
- Boolean
- String
- Opaque Data

The standard also requires that the primitive data types be aligned on their natural boundaries. For example, Float_64 must be aligned on a 64-bit boundary. Address 0 is considered the beginning of the UDP payload; all alignments are relative to Address 0.

The first 4 bytes of the message structure are reserved. After this, the basic message structure consists of a 4-byte word called the Functional Status Set, followed by up to four data sets. The basic message structure can be repeated an arbitrary number of times to form the message structure. Figure 23 depicts two message structures. The one on the left consists of two data sets, *Data Set 1* and *Data Set 2*. The Functional Status Set has two functional status bytes, *FS1* and *FS2*, which correspond to the Data Sets 1 and 2, respectively.

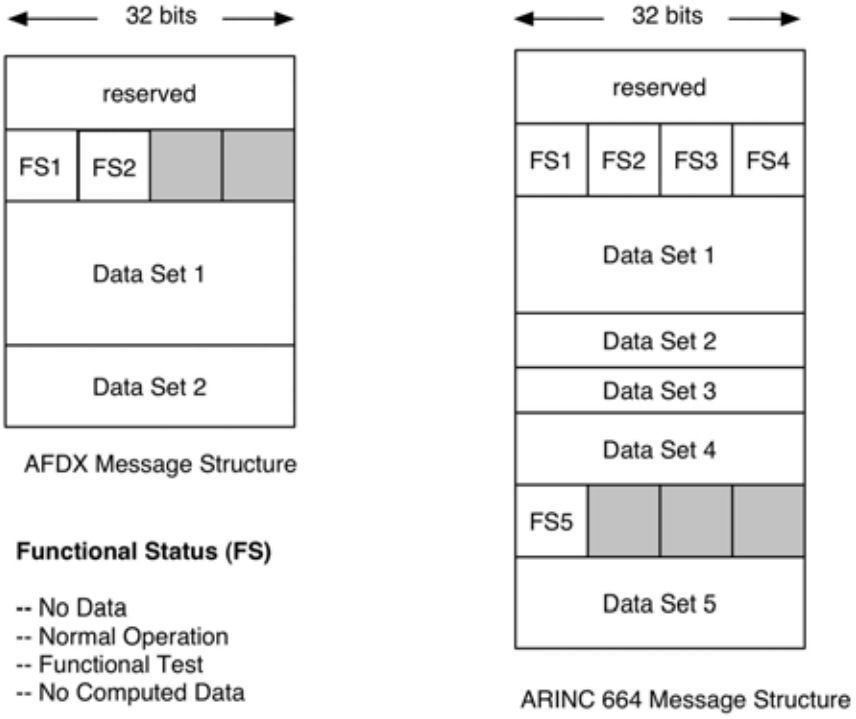

Figure 23. Two Message Structures

The functional status of each data set is encoded in the corresponding Functional Status byte. There are four possible states: No Data, Normal Operation, Functional Test, and No Computed Data. Clearly, the data must be grouped into data sets so that the functional status applies to all the data in the data set.

The message structure depicted above on the right consists of two basic message structures and a total of five data sets and five corresponding functional statuses.

ARINC 429 Labels

Figure 24 shows how ARINC 429 data can be accommodated using the ARINC 664 message structures. The opaque primitive data type is used so that the interpretation of the data is left up to the application (as usual).

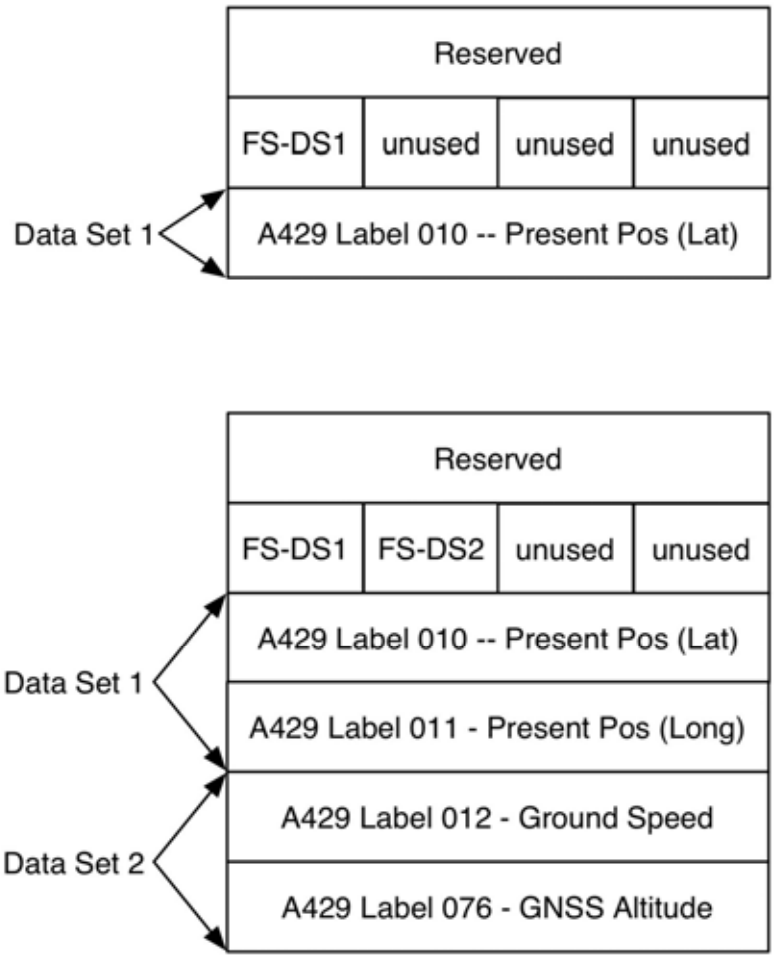

Figure 24. ARINC 664 Message Structures

CHAPTER 12

The AFDX Protocol Stack

The AFDX Protocol Stack

This tutorial concludes with a description of the overall AFDX protocol stacks for transmission and reception. The protocol layers are divided into AFDX communications services, UDP transport layer, and link level (virtual link) services.

Transmission

The Tx protocol begins with a message being sent to an AFDX port. The UDP transport layer is responsible for adding the UDP header, which includes the appropriate source and destination UDP port numbers. These numbers are, in most cases, determined by the system configuration and are fixed for each AFDX communications port. In the case of a SAP port, the application specifies the IP and UDP destination addresses dynamically.

The IP network layer receives the UDP packet and determines whether it needs to be fragmented. The IP network layer uses the appropriate virtual link's Lmax to determine whether fragmentation is necessary. The IP header is added, and IP checksum is calculated for each fragment. The IP layer adds the Ethernet header and enqueues the Ethernet frame in the appropriate sub-VL queue. The (virtual) link layer is responsible for scheduling the Ethernet frames for transmission, adding the sequence numbers (on a per-VL basis), and passing the frames to the Redundancy Management Unit, where the frames are replicated (if necessary) and the Ethernet source address is updated with the physical port ID on which the frame is transmitted.

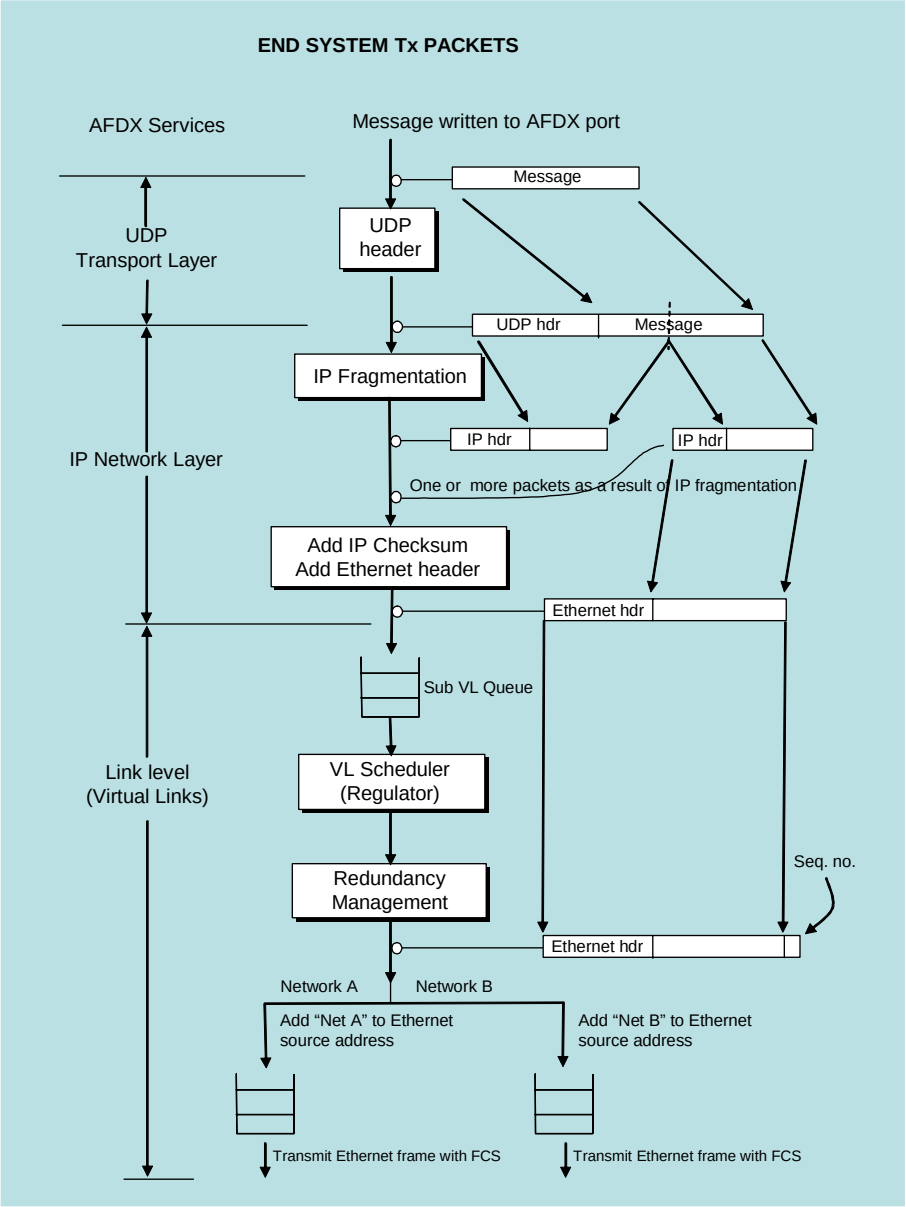

Figure 25. AFDX Tx Protocol Stack

Reception

Reception is the reverse of transmission. The process starts with the reception of an Ethernet frame, which is checked for correctness using the Frame Check Sequence (FCS). If there is no error, the FCS is stripped and the AFDX frame is passed through Integrity Checking and Redundancy Management. These steps are carried out at the (virtual) link level. The resulting IP packet is passed on to the IP network level.

The network level is responsible for checking the IP checksum field and the UDP packet reassembly, if necessary. The UDP packet is passed to the UDP transport layer to deliver (DEMUX) the AFDX message to the appropriate UDP port.

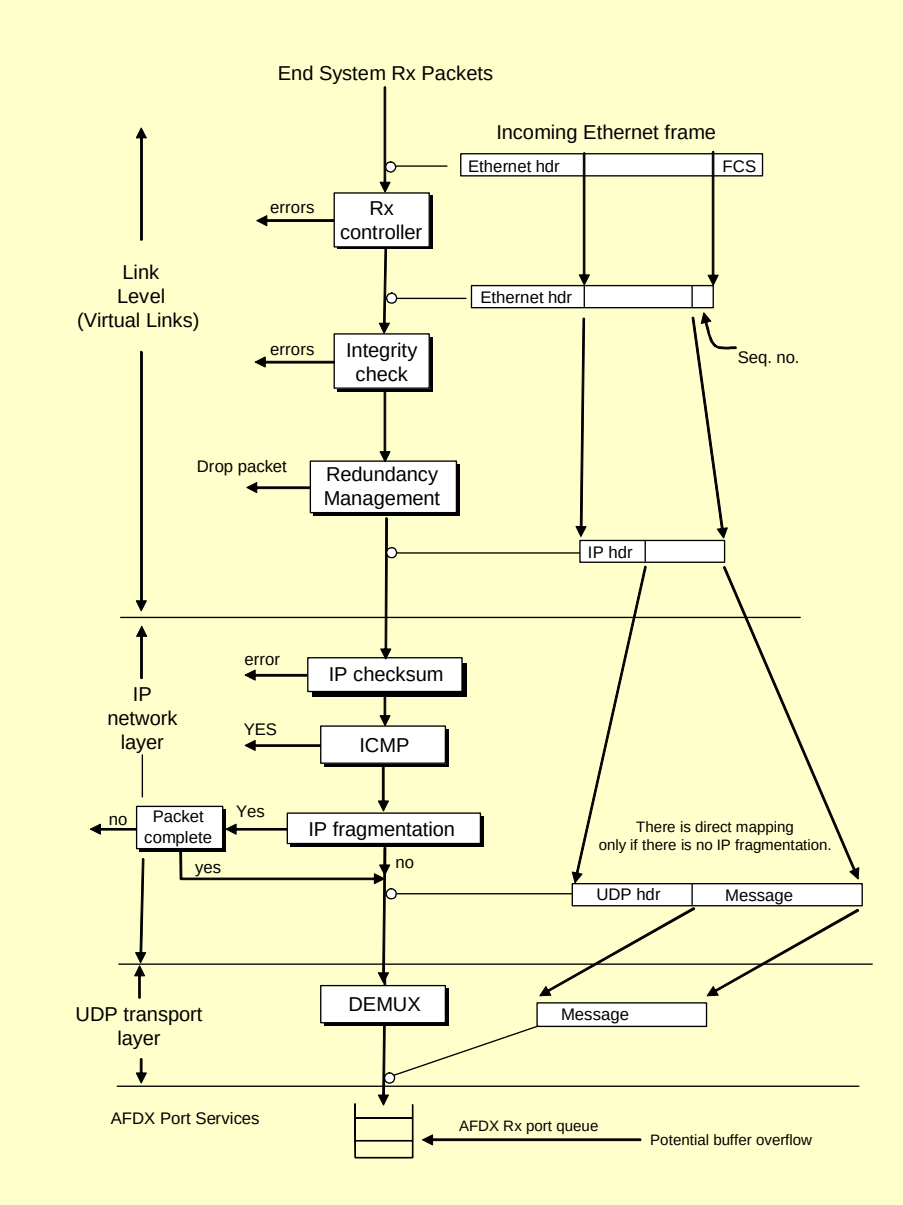

Figure 26. AFDX Rx Protocol Stack

APPENDIX A

AFDX Frame Addressing and Header Structures

Ethernet Addressing

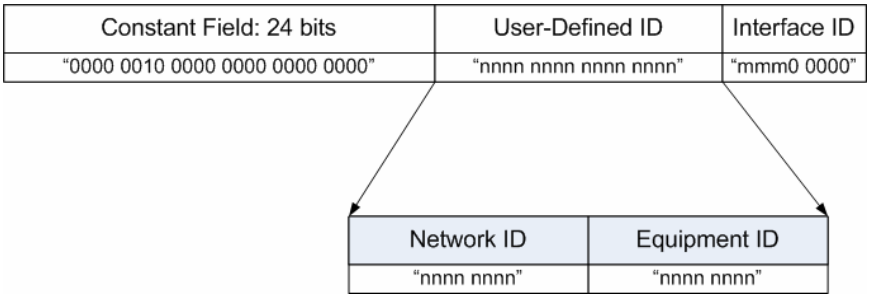

Figure 27. Ethernet Source Address Format

IP Header Format and Addressing

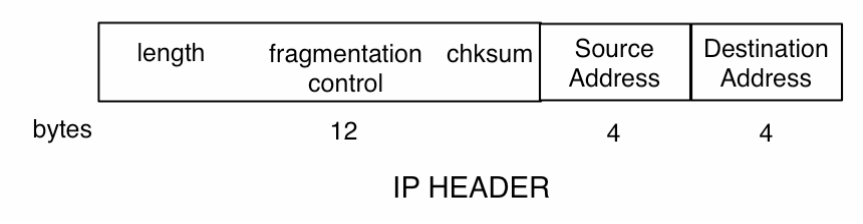

Figure 28. IP Header Format

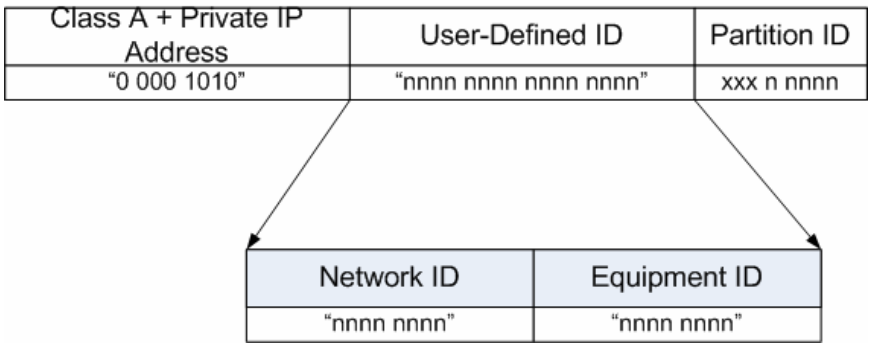

Figure 29. IP Unicast Address Format

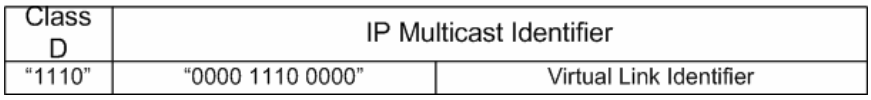

Figure 30. IP Multicast Address Format

UDP Header Format

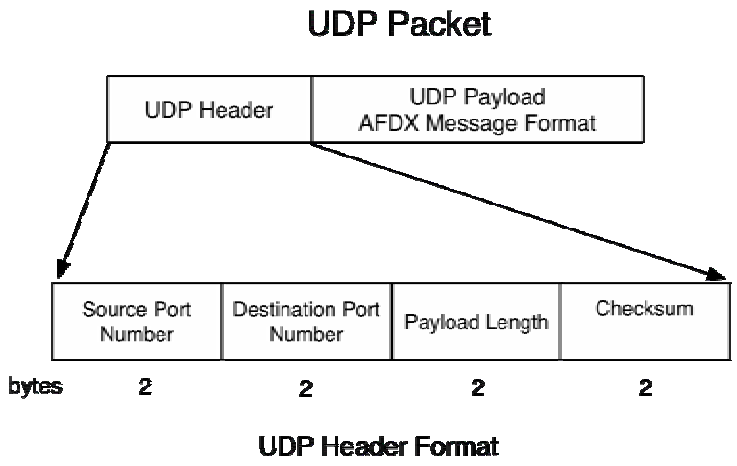

Figure 31. UDP Header Format

APPENDIX B

Referenced Documents

Reference List

Table 2. Referenced Documents

Document Name	Source
ARINC 664, Aircraft Data Network, Part 7 – Avionics Full Duplex Switched Ethernet (AFDX) Network	ARINC 05-005/ADN-39