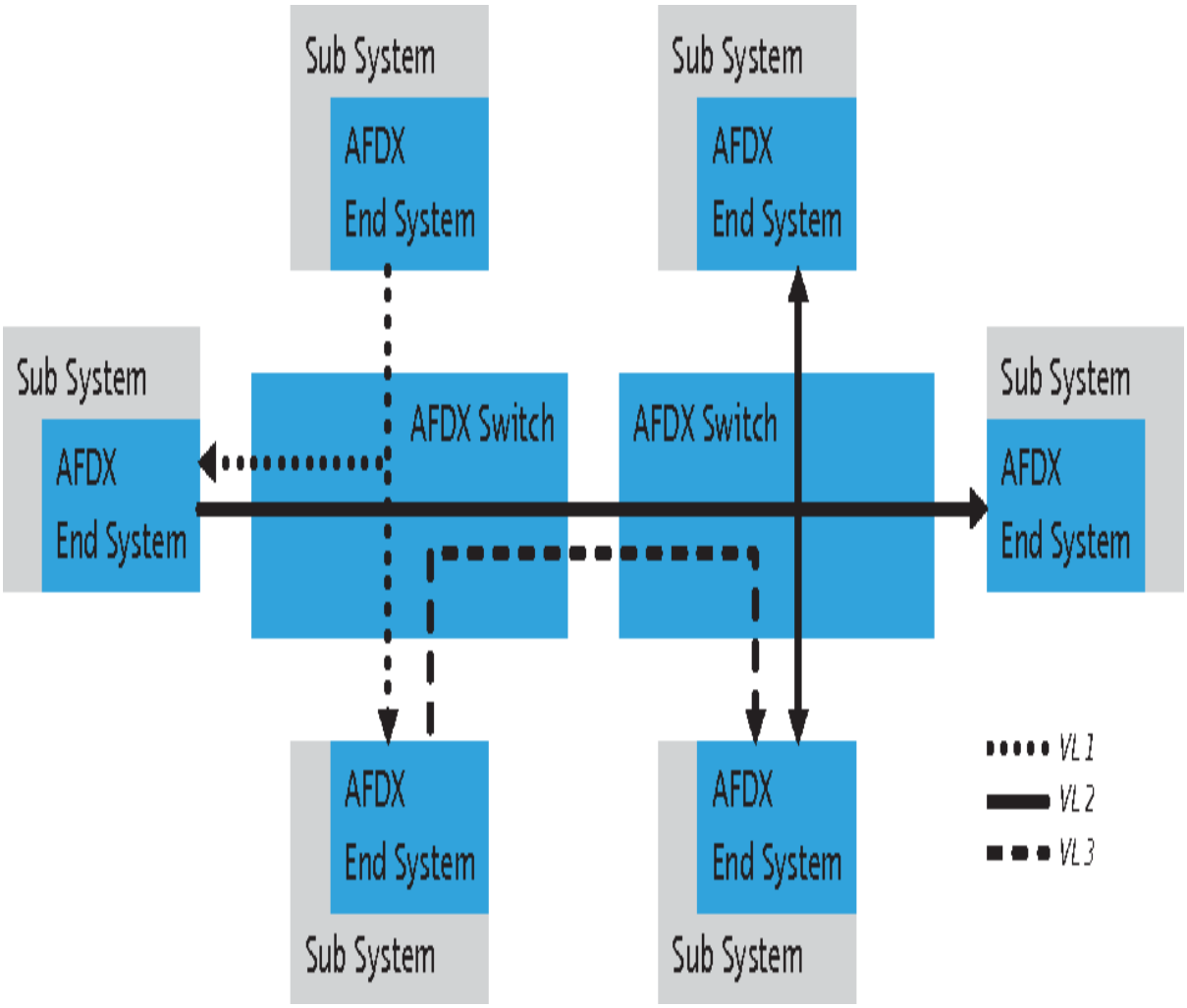
AFDX-PROTOCOL



ARINC 664, also known as Avionics Full-Duplex Switched Ethernet (AFDX), is a standard for a high-speed data network used in modern airplanes. Here's a breakdown of what it is and how it works:

➢ What it is:

- A deterministic Ethernet network designed for safety-critical applications in airplanes.
- Offers guaranteed delivery times for data packets, crucial for functions like flight control.

- Relies on commercially available (COTS) components to keep costs down.

❖ **Technical details:** ARINC 664 is based on commercial off-the-shelf (COTS) Ethernet technology, but with modifications to make it suitable for safety-critical applications in airplanes. It uses a star topology network with switches and end systems.

❖ **Key features:**

❖ **Deterministic:** ARINC 664 guarantees that data will be delivered within a specific time frame, which is crucial for critical systems like flight controls.

❖

❖ **Reliable:** It uses redundancy mechanisms to ensure that data can still be transmitted even if parts of the network fail.**Fast:** ARINC 664 uses high-speed Ethernet links (typically 100 Mbps) to handle the large amount of data generated by modern avionics systems.

❖ **Benefits:**Reduced costs: By using COTS components, ARINC 664 helps to lower the cost of developing and deploying avionics systems.Faster development: The use of standard Ethernet technology allows for faster development of new avionics systems.Increased functionality: ARINC 664 can handle a wider range of data types than older avionics networks, which allows for more complex and integrated avionics systems.

❖ **Applications:** ARINC 664 is used on a variety of modern airplanes, including the Airbus A350, A380, and A400M, as well as the Boeing 787 Dreamliner. It is used for a variety of functions, including flight control, engine control, and cabin management systems.

❖ **Technical details:**

- Based on standard Ethernet technology (IEEE 802.3) but with a specific profile for avionics use.
- Utilizes a star topology network with end systems (avionics devices) connected to switches.
- A key feature is Virtual Links (VLs) - logical data paths defining one-way communication between a source and receivers.
- VLs ensure guaranteed bandwidth and prioritized delivery for critical data.
- The network offers redundancy mechanisms for fault tolerance and continuous operation.

❖Key Features of AFDX:

1. **Full-Duplex Communication:**
   - AFDX uses full-duplex Ethernet, allowing data to be sent and received simultaneously. This improves the efficiency and speed of data transmission.
2. **Deterministic Behavior:**
   - One of the primary enhancements in AFDX over standard Ethernet is the ability to guarantee deterministic data transfer. This means that data is delivered within a known time frame, which is crucial for safety-critical systems in aviation.
3. **Virtual Links (VLs):**
   - AFDX introduces the concept of Virtual Links, which are logical unidirectional data paths between a source and one or more destinations. Each VL has a predefined bandwidth, and traffic is policed to ensure it doesn't exceed this bandwidth, which helps maintain deterministic performance.
4. **Quality of Service (QoS):**
   - AFDX provides different levels of Quality of Service to ensure that critical data is prioritized and delivered on time. This is particularly important in avionics, where data related to flight control and safety must be transmitted reliably.

5. **Redundancy:**
    - To enhance reliability, AFDX networks often use redundancy, with dual networks running in parallel. If one network fails, the other can continue to operate without interruption, ensuring continuous data availability.
6. **Backward Compatibility:**
    - AFDX is compatible with existing Ethernet-based systems, allowing it to be integrated into modern avionics systems without completely overhauling the existing infrastructure.

❖Applications in Avionics:

AFDX is widely used in modern aircraft, such as the Airbus A380 and Boeing 787, where it supports communication between various avionics subsystems, including flight controls, navigation, and in-flight entertainment systems. The protocol ensures that these systems can communicate reliably and in real time, which is essential for both the safety and comfort of passengers.

Standards:

AFDX is standardized under ARINC 664, which is a part of the ARINC 600 series of standards. This standard outlines the specifications for data networks in avionics, ensuring compatibility and interoperability between different systems and manufacturers.

In summary, AFDX is a critical technology in modern avionics, providing a robust, deterministic, and reliable networking solution that meets the demanding requirements of aircraft systems.

In AFDX (Avionics Full-Duplex Switched Ethernet), the data frame structure is based on the standard Ethernet frame but is tailored to meet the specific needs of avionics systems. Below is an outline of the AFDX data frame structure:

AFDX Data Frame Structure

1. **Preamble (7 bytes)**

- The preamble consists of a sequence of bits used to synchronize the receiver's clock with the incoming data stream. It helps the receiving device recognize the start of the frame.

2. Start Frame Delimiter (SFD) (1 byte)

- This byte indicates the start of the Ethernet frame and signals that the actual data follows.

3. **Destination MAC Address (6 bytes)**

- The destination MAC address specifies the intended recipient of the frame. In an AFDX network, this could be a specific avionics device or a multicast address.

4. **Source MAC Address (6 bytes)**

- The source MAC address indicates the origin of the frame. It identifies the device that is transmitting the data.

5. **EtherType/Length (2 bytes)**

- This field can either specify the length of the payload or indicate the protocol type used in the data field. In AFDX, it usually signifies that the frame carries an IP packet or some other specific protocol

6. **Virtual Link Identifier (VLID) (2 bytes)**

- The VLID is a key AFDX-specific addition. It identifies the Virtual Link (VL) that this frame belongs to. Each VL has a unique ID, ensuring that the data is routed correctly within the AFDX network.

7. **Payload (46-1500 bytes)**

- The payload contains the actual data being transmitted. The size can vary depending on the application, but it must conform to the minimum and maximum Ethernet frame sizes.

8. **Frame Check Sequence (FCS) (4 bytes)**

- The FCS is a checksum calculated over the frame's content. It is used for error detection to ensure that the frame has not been corrupted during transmission.

9. **Interframe Gap (IFG) (12 bytes)**

- The IFG is not part of the frame itself but is a mandatory idle period between two consecutive Ethernet frames. It

allows time for the receivers to process the current frame before the next one arrives.

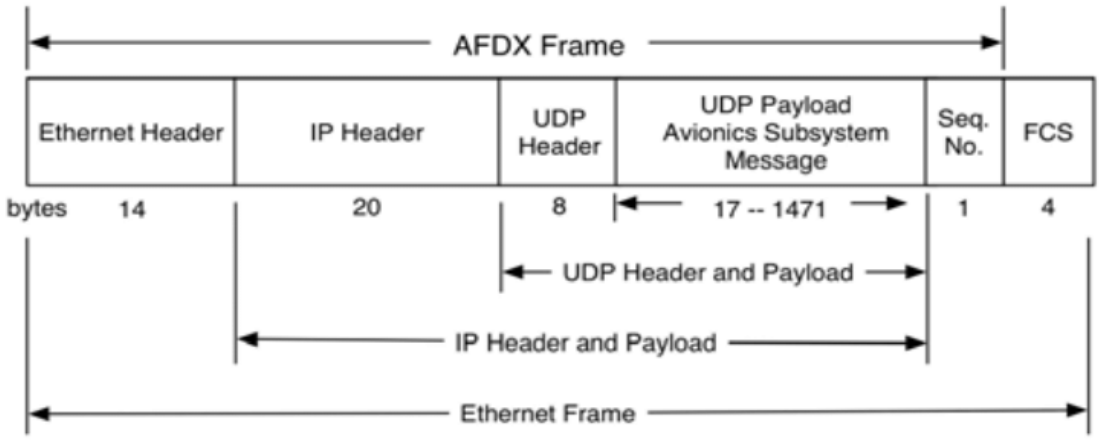
Example Breakdown:

1. **Preamble:** 7 bytes of alternating 1s and 0s.
2. **SFD:** 1 byte, usually 10101011 in binary.
3. **Destination MAC Address:** 6 bytes, unique hardware address of the destination.
4. **Source MAC Address:** 6 bytes, unique hardware address of the source.
5. **EtherType/Length:** 2 bytes, typically 0x0800 for IP.
6. **VLID:** 2 bytes, specific to the AFDX protocol.
7. **Payload:** Variable length, containing the data.
8. **FCS:** 4 bytes, used for error-checking.

Special Considerations in AFDX:

- **Deterministic Data Transfer:** AFDX ensures that frames are transmitted within a guaranteed time, which is critical for real-time avionics applications.
- **Redundancy:** AFDX frames may be duplicated across two independent networks to ensure fault tolerance.

This structure allows AFDX to provide the high reliability and deterministic performance needed for avionics systems while remaining compatible with standard Ethernet technologies.



AFDX DATA FRAME:

7 bytes	1 byte	6 bytes	6 bytes	2 bytes	20 bytes	8 bytes	17 to 1471 bytes	1 byte	4 bytes	12 bytes
Preamble	Start Frame Delimiter	Destination Address	Source Address	Type IPv4	IP Structure	UDP Structure	AFDX Payload	Seq Number	Frame Check Seq	Interframe Gap

DATA TRANSMISSION AND RECEPTION :

The implementation of **AFDX (Avionics Full-Duplex Switched Ethernet)** data transmission and reception in **C** is typically built on top of a custom **AFDX stack** or uses specialized AFDX libraries and hardware that handle communication at the protocol level. AFDX is a high-level protocol with additional features such as **redundancy**, **determinism**, and **traffic shaping** based on **virtual links**.

General Steps for AFDX Data Transmission and Reception:

1. **Initialize the AFDX communication system**.
2. **Configure Virtual Links (VLs)** for specific communication channels.
3. **Transmit data** over a specified virtual link.
4. **Receive data** from a virtual link.
5. **Handle error checking** and **timeouts** (as required in AFDX).

Example C Code for AFDX Data Transmission and Reception

The following is a conceptual example of how you might structure a C program for **AFDX data transmission and reception**. This assumes that you are using a library that provides a basic API for AFDX.

Header Includes:

```
#include <stdio.h>

#include <stdlib.h>

#include <string.h>

// Hypothetical AFDX Library API

#include "afdx_lib.h"  // Assume this provides the necessary AFDX stack functions
```

//Define Constants:

```
#define VIRTUAL_LINK_ID_TX 1001   // Virtual Link ID for transmission

#define VIRTUAL_LINK_ID_RX 1002   // Virtual Link ID for reception

#define AFDX_BUFFER_SIZE  1024    // Buffer size for data transmission/reception

// Initialization and Configuration:

// AFDX Initialization

void init_afdx() {

    if (afdx_init() != 0) {

        printf("AFDX initialization failed\n");

        exit(EXIT_FAILURE);

    }

    printf("AFDX initialized successfully\n");

    // Configure Virtual Link (VL) for transmission

    if (afdx_configure_vl(VIRTUAL_LINK_ID_TX) != 0) {

        printf("Failed to configure VL for transmission\n");
```

```
        exit(EXIT_FAILURE);
    }
    // Configure Virtual Link (VL) for reception
    if (afdx_configure_vl(VIRTUAL_LINK_ID_RX) != 0) {
        printf("Failed to configure VL for reception\n");
        exit(EXIT_FAILURE);
    }
    printf("Virtual Links configured successfully\n");
}
```

Data Transmission (Sending Data Over AFDX):

```
// AFDX Data Transmission
void send_afdx_data(const char* data, size_t
data_size) {
    if (data_size > AFDX_BUFFER_SIZE) {
        printf("Data size exceeds buffer
limit\n");
        return;
    }

    int status = afdx_send(VIRTUAL_LINK_ID_TX, data, data_size);
    if (status == 0) {
        printf("Data transmitted successfully over VL %d\n",
VIRTUAL_LINK_ID_TX);
    } else {
        printf("Data transmission failed\n");
    }
```

```
}
```

Data Reception (Receiving Data Over AFDX):

```
// AFDX Data Reception
void receive_afdx_data() {
    char buffer[AFDX_BUFFER_SIZE];
    int received_size = afdx_receive(VIRTUAL_LINK_ID_RX, buffer, AFDX_BUFFER_SIZE);
    if (received_size > 0) {
        printf("Received data over VL %d: %s\n", VIRTUAL_LINK_ID_RX, buffer);
    } else {
        printf("No data received or reception error\n");
    }
}
```

Main Function to Demonstrate Transmission and Reception:

```
int main() {
    // Step 1: Initialize AFDX Communication
    init_afdx();
    // Step 2: Prepare Data for Transmission
    const char* data_to_send = "Test AFDX Data Transmission";
    // Step 3: Send Data over AFDX
```

```
    send_afdx_data(data_to_send, strlen(data_to_send));

    // Step 4: Receive Data from AFDX

    receive_afdx_data();

    // Step 5: Shutdown AFDX

    afdx_shutdown();

    printf("AFDX communication ended\n");

    return 0;
```

}

Explanation:

1. **Initialization (`init_afdx`)**:
   - Initializes the AFDX stack and configures two virtual links (VLs). One VL is used for transmission (`VIRTUAL_LINK_ID_TX`), and the other is for reception (`VIRTUAL_LINK_ID_RX`).
2. **Data Transmission (`send_afdx_data`)**:
   - Sends data over a specific virtual link using a fictional `afdx_send()` function provided by the AFDX library. The `afdx_send()` function is assumed to handle packet fragmentation, redundancy, and QoS according to AFDX protocol.
3. **Data Reception (`receive_afdx_data`)**:
   - Receives data from a specified virtual link using `afdx_receive()`. The received data is stored in a buffer and then printed to the console. The size of the received data is checked to ensure proper reception.
4. **Main Function**:
   - Demonstrates the overall flow by initializing the AFDX system, sending some data, and then attempting to receive data.

AFDX Library Functions (Assumed):

`afdx_init()`: Initializes the AFDX system.

`afdx_configure_vl()`: Configures the virtual links.

`afdx_send()`: Sends data on a specific virtual link.

`afdx_receive()`: Receives data from a specific virtual link.

`afdx_shutdown()`: Shuts down the AFDX system.

Real-World Considerations:

- **AFDX Libraries and Hardware**: In actual AFDX implementations, you'll likely be using vendor-specific AFDX APIs (e.g., Safran, TTTech) and need to integrate the low-level hardware drivers for communication.
- **Error Handling**: A real implementation would involve more detailed error handling, including timeouts, redundancy management, and retries as required by the AFDX specification.
- **Determinism**: The AFDX protocol guarantees bounded latency and determinism, so the actual implementation needs to follow strict timing and prioritization rules.