Implementation Issues in Networked Embedded Control Systems

Embedded Networks and Real-Time Scheduling

FlexRay, AFDX and ARINC 653 examples

References/Copyright

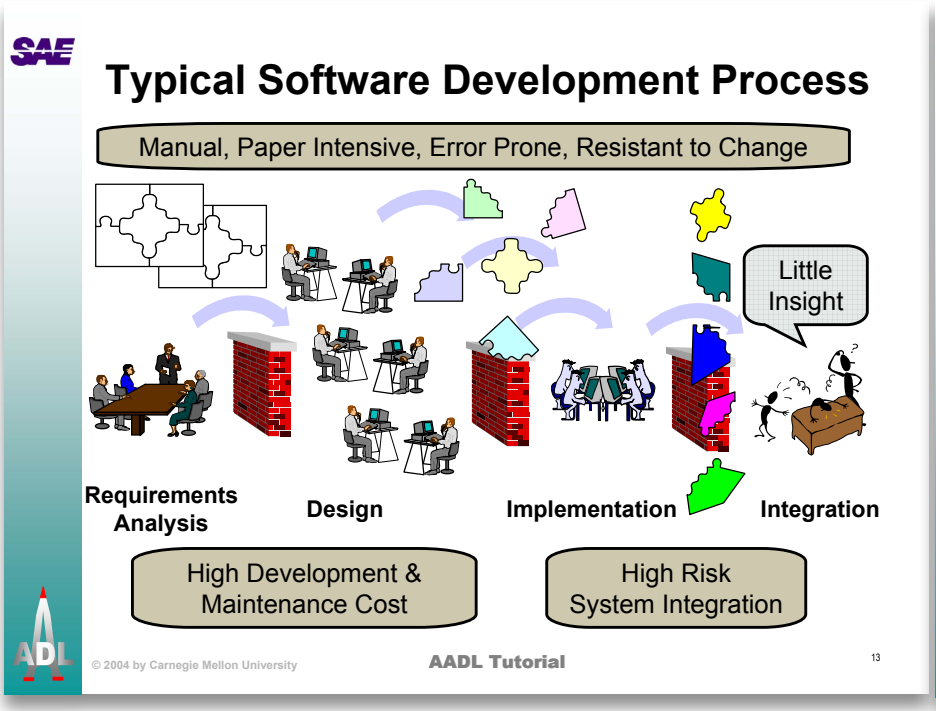
- FlexRay material from National Instruments tutorial
- AFDX/ARINC material and figures taken from GE Intelligent Platforms, “AFDX/ARINC 664 tutorial”

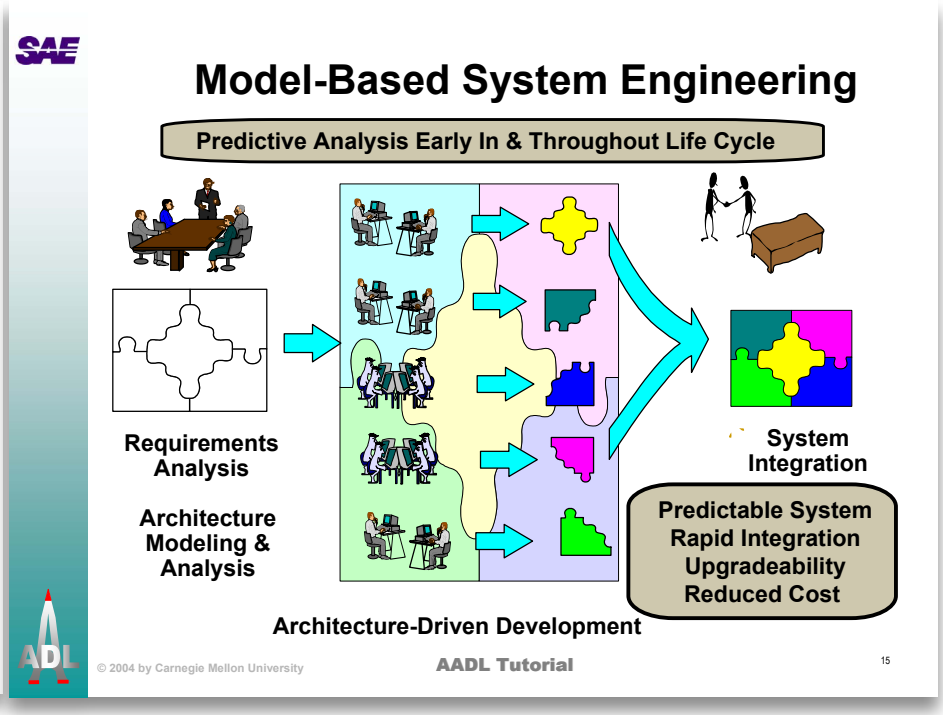
Goal

- Discuss current implementation practices for real-time networked embedded systems, specifically for digital control applications.

- Underlying motivation:
  - better modeling and understanding of impact of implementation on control algorithms. Develop more realistic analysis of the behavior of modern sampled-data systems.
  - more robust/dependable/convenient/efficient initial control designs. Reduce the gap between control design stage and implementation stage.

Desired Development Process

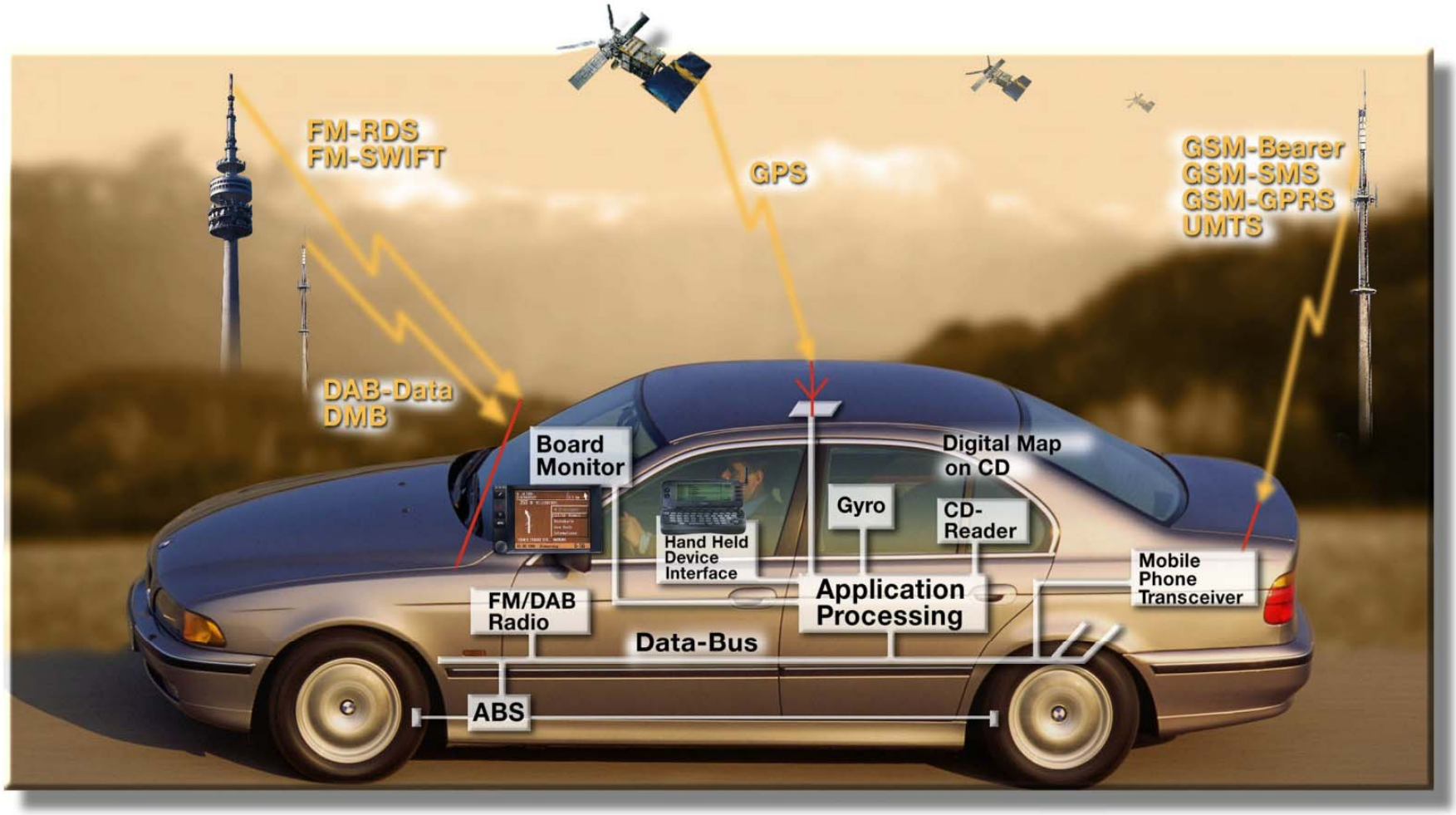




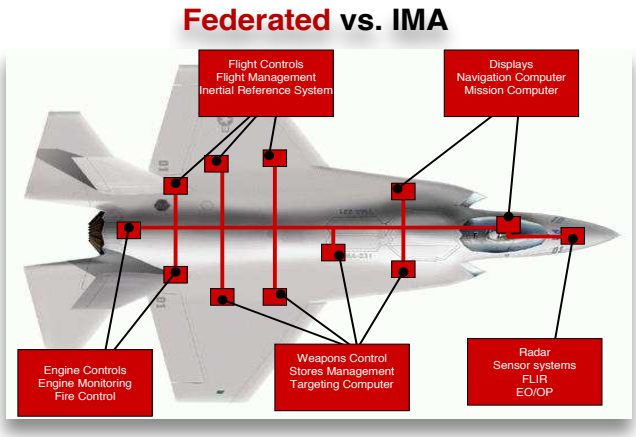
[AADL presentation: Bruce Lewis and Peter Feiler]

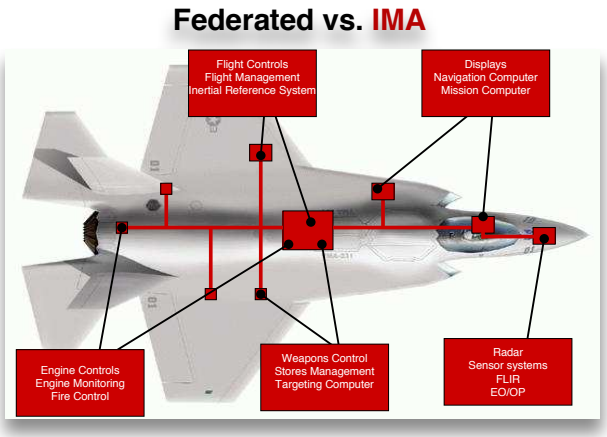
- Holy grail of Model-Based Engineering, essentially...
- Probably not achievable without rethinking the interfaces between the disciplines and using domain specific knowledge. This in turns require the knowledge of several disciplines (e.g. control design and real-time system programming).

Automotive Electronics

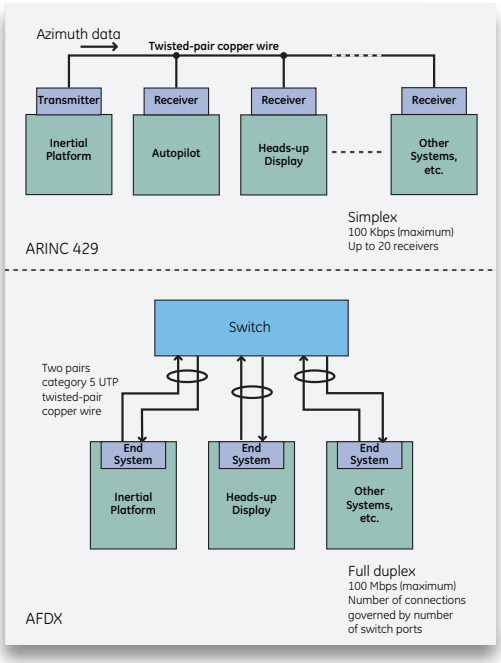


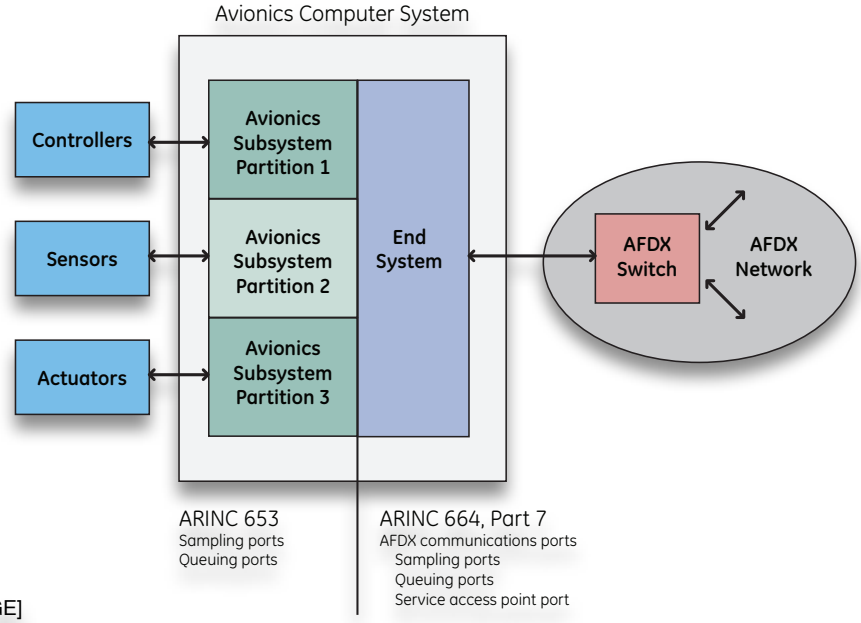
Avionics





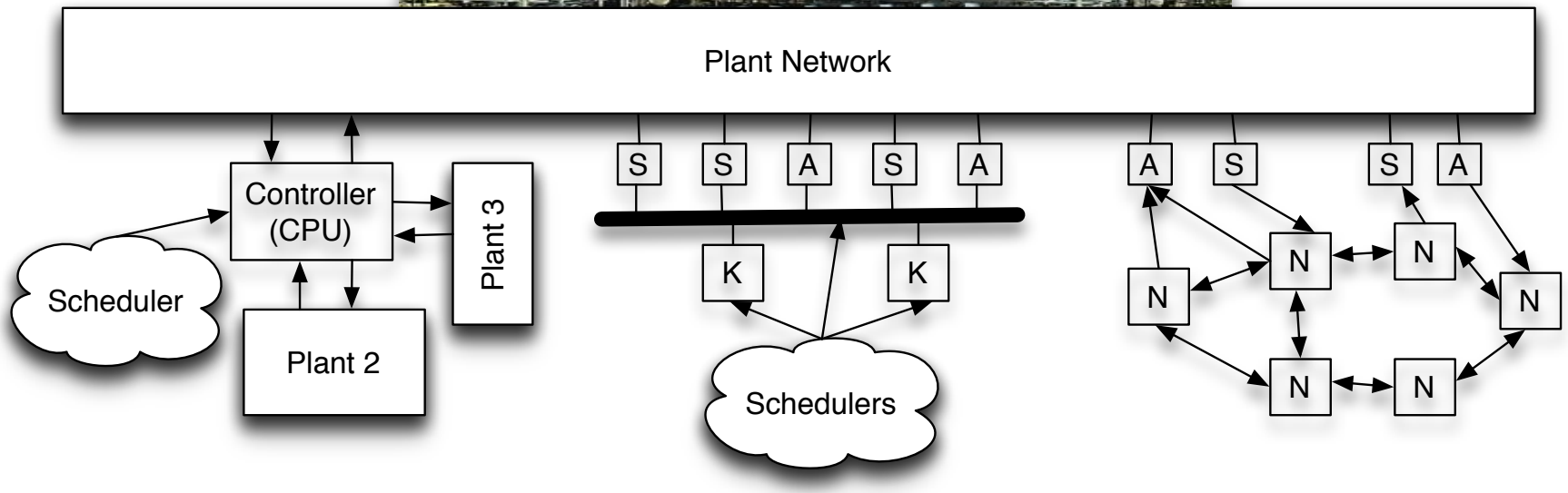
[© Wind River]





[© GE]

Variety of Control System Architectures



So far: single processor directly connected to the plant and dedicated to the control task

We need to improve our sample-data system models to rigorously and efficiently design control loops on such systems (e.g. simplify system integration)

For this, we must know a little bit about resource management practice for networks and processors

Digital Control Issues in Complex Embedded Systems

- Various components of the control loops are now connected via communication network
  - communication delays, lost messages
- The communication medium tends to be shared by several applications
  - need management of communication resources
- The CPUs are not necessarily dedicated to the control tasks, but shared by other tasks, safety critical and non safety critical
  - need management of computation resources, typically by Real-Time Operating System (RTOS) scheduler
- The system design choices for communication and computation (hardware, protocols, etc.) are not necessarily dictated by the control engineers !! (cost, in many industries, is a big factor; power, ...)
  - E.g. CAN networks in automotive application introduce time-varying delays, bad from control perspective. Still, control and other timing concerns tends to slowly influence choices (TT-CAN, FlexRay, AFDX...).
  - Desire to use Commercial Off-the-shelf (COTS) components, reuse components and software (e.g. OS) from other/previous systems.

Design Approaches and Questions

- Design control loops as usual, ignoring implementation issues
  - Research question: analyze the impact of implementation choices on performance
- Control and Embedded System Co-design: design control loops together with the scheduling policies, communication protocols, etc.
  - Popular research topic, probably often not a viable possibility due to previously mentioned design constraints. Can try to implement solution on top of given protocol (e.g. control server scheduled periodically by RTOS).
- Design implementation aware control loops.
  - Add compensating mechanisms within the control laws, for delays, jitter, packets losses.
  - Control laws more complex to implement, but we are (as much as possible) only modifying the control systems, not the global system design.
  - Expect potential loss in performance wrt co-design, but should be better than ignoring issues. Also, in this approach, we would like to allow various possible choices of implementations at a later system design stage: leave more freedom to the systems engineer, allow for subsequent changes, increase possibilities of component reuse, etc.

Control Networks

Communication Network

- Initially point-to-point communications between all subsystems of a control loop
- Replaced now usually by wired data bus (with or without bus controller); sometimes switched networks (AFDX)
- Some interest in wireless networks for industrial applications (e.g. process control) – still controlled environment however, not ad-hoc networks...
- Delays, reliability, jitter, etc. are network and protocol dependent. Impacts on performance of the control system must typically be understood.

Automotive Industry Examples (bus technologies)

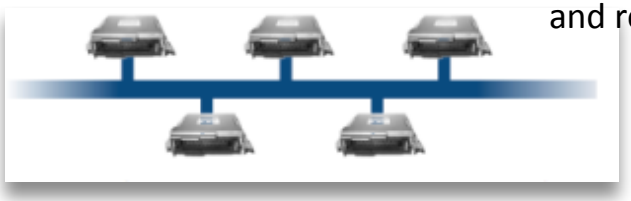
- De facto Standard in Automotive Industry: CAN (Controller Area Network), first version released by Bosch in 1986, first chips by Intel and Philips in 1987.
- CAN enables communication between ECUs, actuators and sensors: e.g. engine control unit, transmission, airbags, ABS, cruise control subsystems; control loops might share the same bus.
- Medium cost → used now in other industrial distributed control applications (fieldbus). Speed: 1 Mbit/s for up to 40m.
- If the bus is free, any node may begin to transmit. Message with dominant id overwrites others in case of collision. Non-determinism, time-varying delays are potential issues for control applications. Does not guarantee timely delivery of data nor very high rates.
- Another ex. of protocol for control systems: TTP (Time-Triggered Protocol), originally designed at the Vienna University of Technology in the early 80s (Kopetz et al.). TDMA = Time Division Multiple Access → timing more predictable, minimal jitter. Necessary clock synchronization. TDMA can also waste bandwidth if traffic not predictable.
- FlexRay: time- and event-triggered behavior, to accommodate different types of traffic (periodic and sporadic). Developed between 2000 and 2009 by companies in the automotive industry (mostly BMW) as possible future direction. Higher data rates (up to 10 Mbit/s), but currently more expensive → currently for high-end applications.

Bus	LIN	CAN	FlexRay
Speed	40 kbit/s	1 Mbit/s	10 Mbit/s
Cost	$	$$	$$$
Wires	1	2	2 or 4
Typical Applications	Body Electronics (Mirrors, Power Seats, Accesories)	Powertrain (Engine, Transmission, ABS)	High-Performance Powertrain, Safety (Drive-by-wire, active suspension, adaptive cruise control)

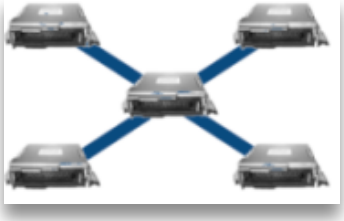
FlexRay

Network Topologies

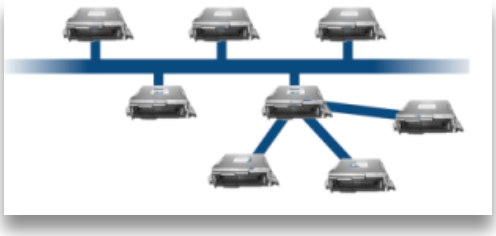
Multi-drop Bus: like CAN. Simplify evolution, lower cost



Star Network: longer distances, network segmentation possible for fault isolation, better performance (e.g. noise integrity) and reliability



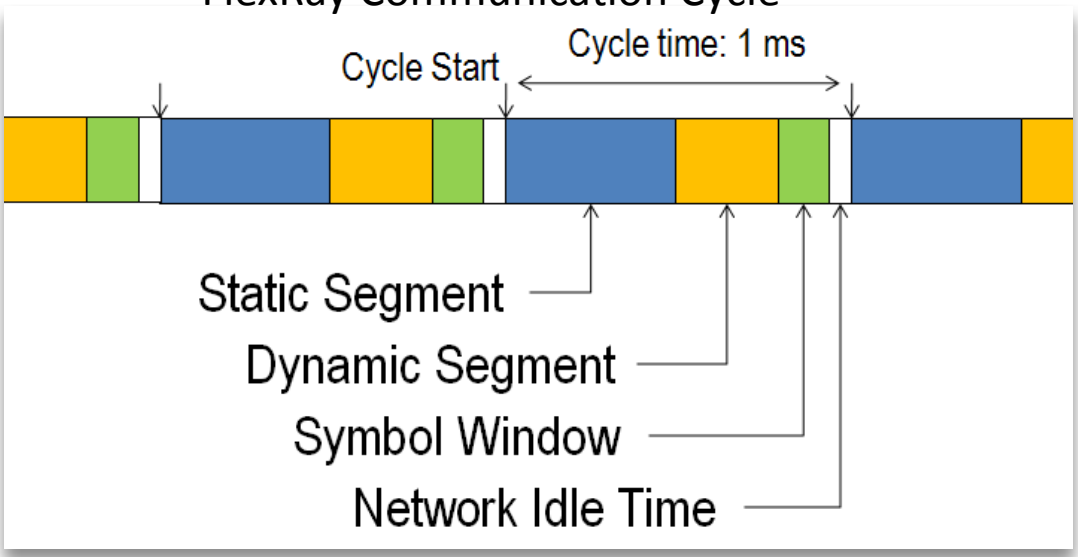
Hybrid Networks



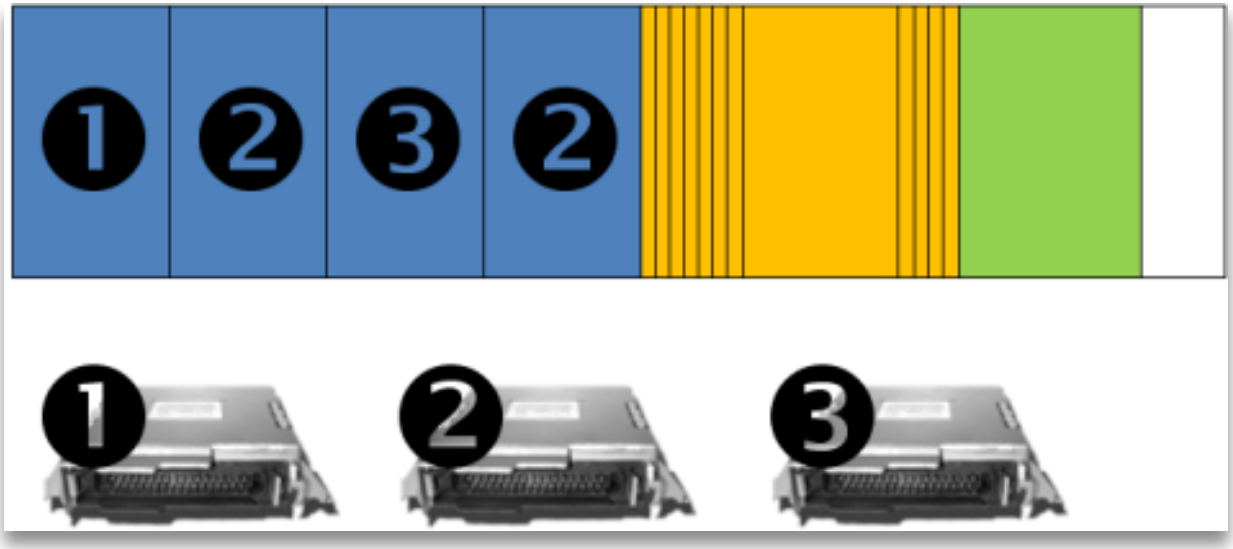
- Pre-set communication cycle, typically around 1-5 ms
- Time-triggered segment for deterministic data arriving in predictable time frame
- CAN-like segment for dynamic event-driven data
- Length of each segment is a design parameter
- *Symbol window*: for network maintenance and start
- *Idle Time*: to maintain synchronization between node clocks. Synchr. to ~1µs

low-level

FlexRay Communication Cycle



Static Segment



- TT segment is very useful for control loops. Gives consistently spaced data.
- If an ECU does not transmit data in one of its slots, it is not used.
- Each slot of a static or dynamic segment contains a FlexRay frame. Length of payload of a frame is 0...254 bytes (>30 times CAN frame payload).
- Synchronization using Sync frames broadcasted by pre-designated nodes.

In-cycle control

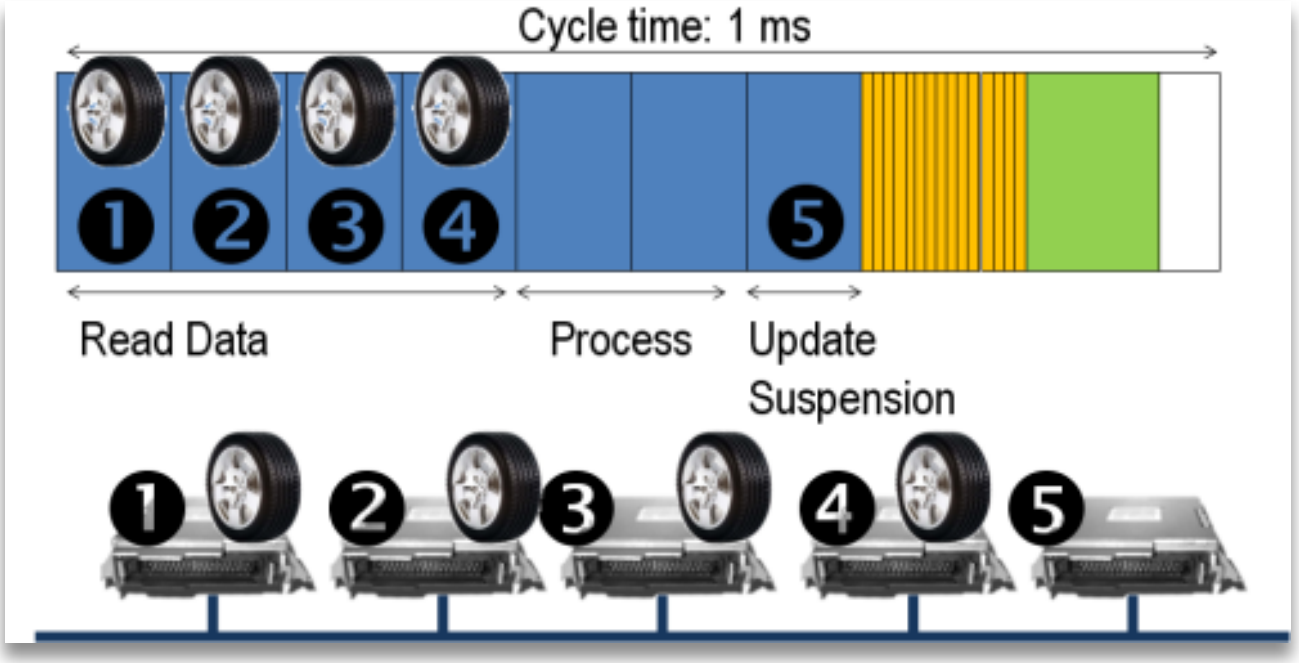
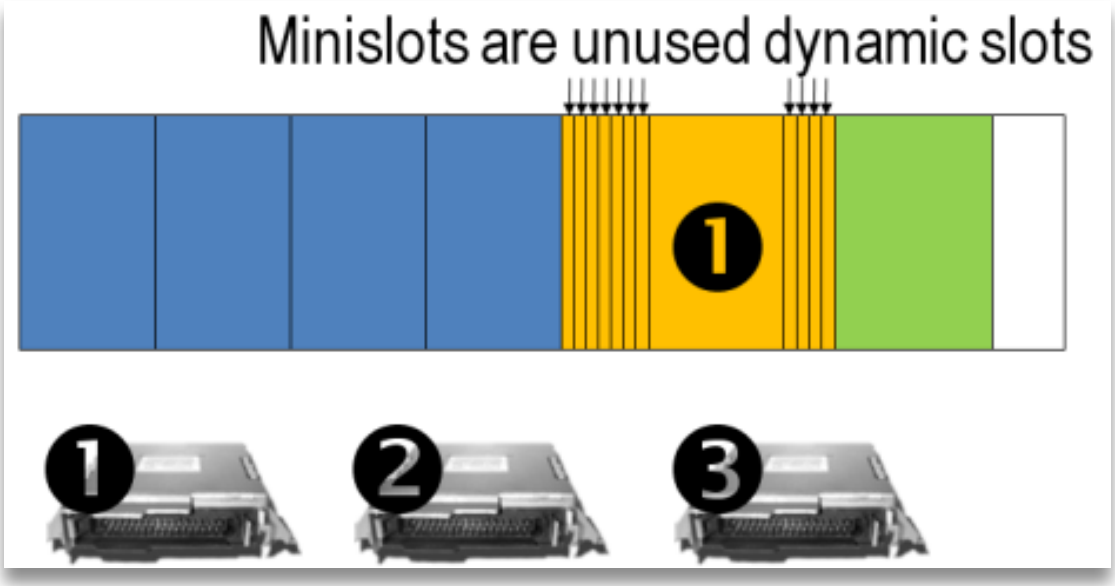


Figure 13. In-cycle control reading 4 wheel positions and updating a vehicle control output in a single FlexRay cycle.

- Possibility of high-speed control rates: can read sensors and update control input within same cycle

Dynamic Segment



- For occasionally transmitted data
- Fixed length
- Higher priority data eligible for transmission in dynamic segment receive a minislot closer to the beginning of the dynamic frame
- If ECU doesn't broadcast in its minislot, it loses its spot. Process continues until and ECU starts broadcasting. End-result similar to CAN arbitration.

Some Differences between Embedded Networks and other Networks

- Embedded networks: closed configuration, does not change one assembled.
  - Eliminates need for some mechanisms like discover and configure devices at run-time
  - Reduces cost, increases reliability of network
- With FlexRay network (and TDMA more generally), each participating node must be configured correctly and know all the parameters of the network.
  - higher implementation complexity and cost than CAN, but better features for control

Network Delays

- Delays due to resolution of resource contention (two mechanisms discussed before)
- Size of individual network reservation slots determined by size of frame
  - CAN high-speed at 1 Mbit/s frame (bus length < 40m). 1 bit = 1 µs
  - CAN extended frame ~128-134 bits. Variable due to bit stuffing. → frame duration is ~ 134 µs.
- Some time reserved for network maintenance (e.g. synchronization)
- Propagation delay:
  - obviously depends on (in fact imposes limit on) the bus length
  - ~ 5 ns/m typical for twisted-pair line of CAN (~3.33 ns/m is speed of light)
- Conclusions:
  1. For many embedded control applications over short networks, e.g. automotive applications: delays mostly due to resource contention. Need to wait for other nodes to finish transmitting. Can optimize sampling times to minimize some of these delays, in particular with TT protocols.
  2. When the network length increases, propagation delay can become an issue. Typical control applications are in teleoperation (e.g. robotic surgery or exploration). E.g. control of a robotic arm in Washington over the internet, with surgeon in California: round-trip delay is ~60-75ms and highly variable (TCP/IP not really made for control...).
  3. For unreliable networks (non-negligible drop rate), can trade-off transmission probability for transmission delay (as in UDP vs. TCP).

Network Delay Models for Control

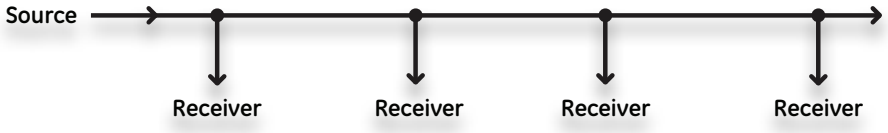
- In my opinion, in the literature on networked control systems, it is too often left unclear what the targeted applications and corresponding important sources of delays are. Would benefit by stating more precise and realistic network properties.
- Propagation delay tends to be emphasized
  - Due to extensive literature and many mathematical results on delay in control, applying to NECS seems convenient
  - Relevant mostly for large networks: multi-robot communications, control of the electric grid, ...
  - In practice, in particular for teleoperation, special techniques have been introduced somewhat independently to deal with large network delays and variability
    - For teleoperation, based on passivity and “wave variables”: Anderson and Spong [88], Niemeyer and Slotine [91]. Mostly for high transmission probability (TCP rather than UDP).
- For many NECS applications with short networks, choices of models seems debatable. E.g.: SISO systems or MIMO with all sensors communicating simultaneously (single frame?) yet subject to large network delays
  - presumably propagation delay?
  - or very long frame (containing many measurements)?
  - or high load already on the network? realistic for control network?
  - Why would the sensors communicate all at once? In practice many sensors with different sampling rates (e.g. IMU at 1000 Hz vs. GPS at 1 Hz)

Avionics Example: AFDX (switch technology)

- All electronic fly-by-wire now only control system used on new airliners
- Other on-board safety-critical system systems rely on timely delivery of data: communications, inertial platforms, etc.
- "Avionics Full-DupleX, switched Ethernet". Original concept by Airbus. Evolved into a standard: IEEE 802.3 (Ethernet frame format) and ARINC 664, Part 7.
- Used by Airbus for A380
- Similar system (ARINC 629) used by Boeing for 777 Dreamliner
- Replaces ARINC 429 (point-to-point technology), and also MIL-STD 1553 (bus technology). Higher data rate: 100 Mbps, 1000 times faster than ARINC 429.

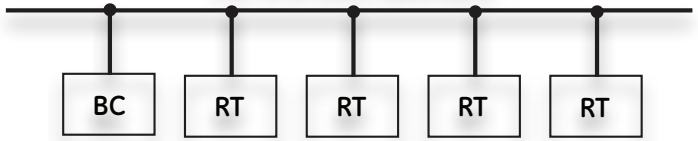
Avionics Networks

ARINC 429



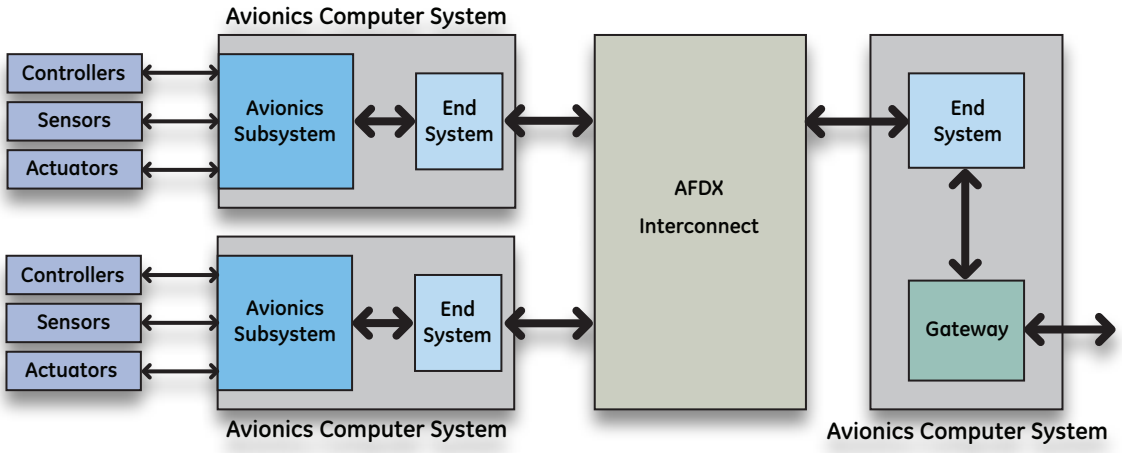
Bit rates are either 100 Kbps or 12.5 Kbps
32-bit messages

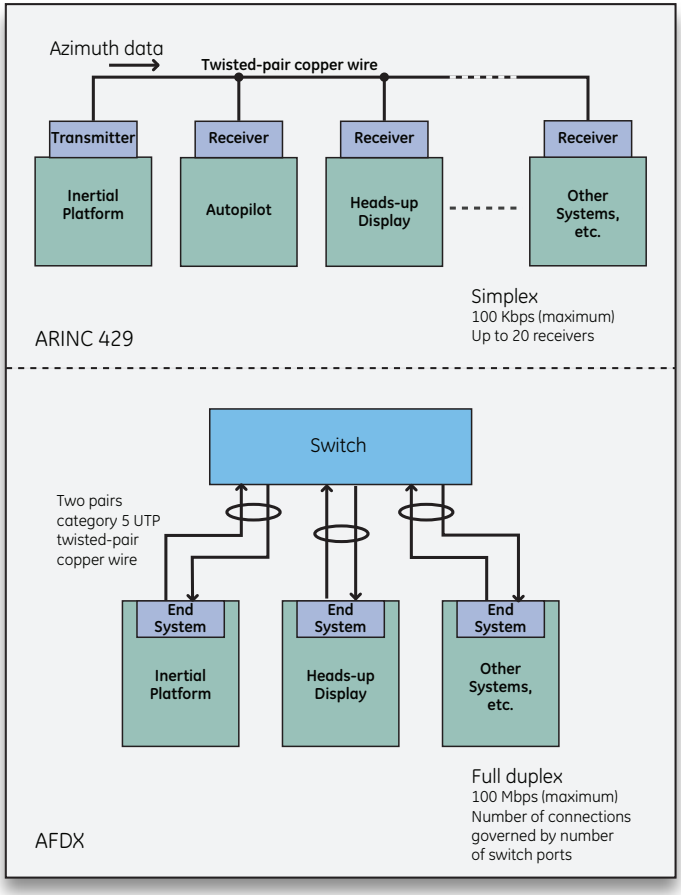
MIL-STD 1553 DATA BUS



Bit-rate 1 Mbps
20-bit data word

AFDX





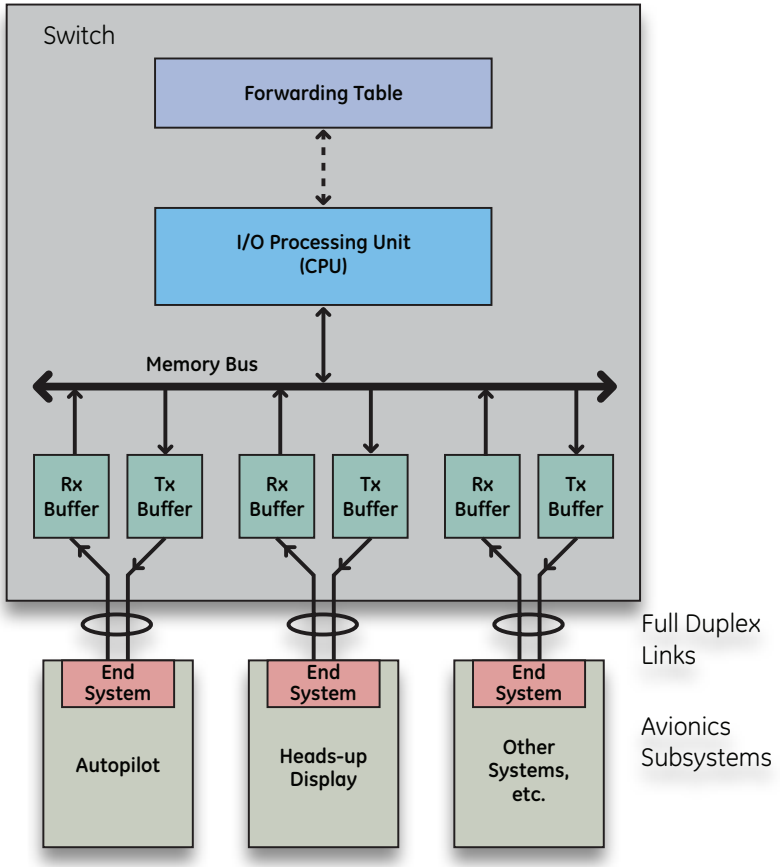
AFDX is Based on Ethernet

- benefits from many investments and advancements since 1972
- Ethernet has no centralized bus control: Transmissions can collide. “CSMA/CD” protocol (Carrier Sense, Multiple Access, and Collision Detection)
  - If you have a message to send and the medium is idle, send the message.
  - If the message collides with another transmission, try sending the message later using a suitable back-off strategy → non-deterministic behavior, possibility of repeated collisions
- Ethernet frame between 64 and 1518 bytes
- Ethernet comm. is connectionless. ACK must be handled at higher levels in the protocol stack

AFDX: Full-duplex, Switched Ethernet

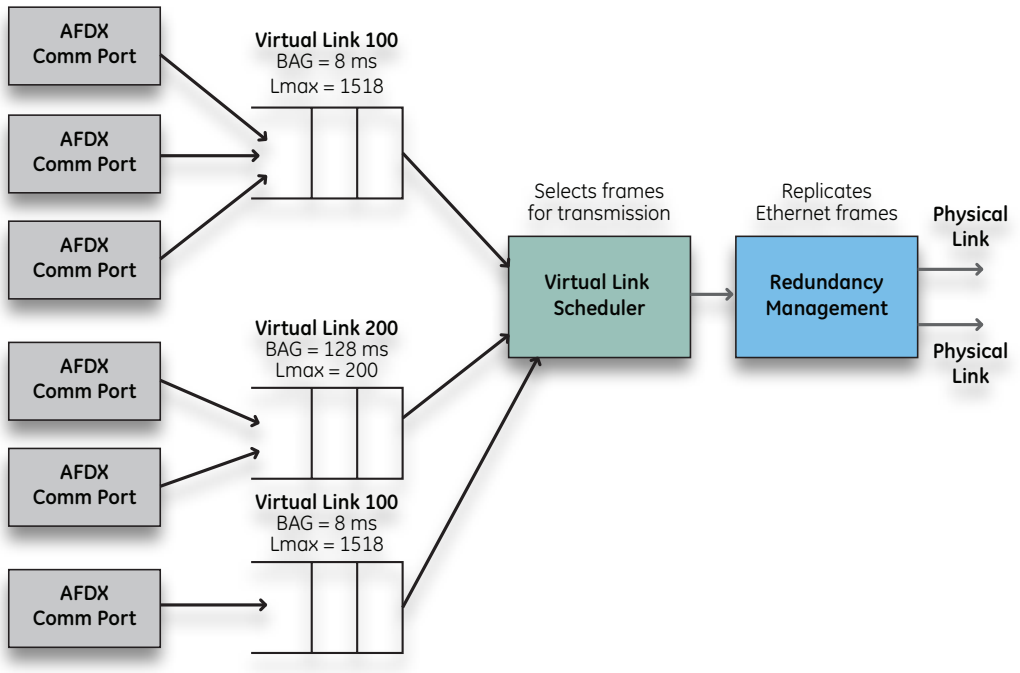
- With Ethernet, very large transmission delays are theoretically possible
- AFDX bounds the max. transmission time between a Tx and a Rx
- Moves from Half-duplex Ethernet to Full-duplex Switched Ethernet to eliminate possibility of collisions
- Now switch needs to move packets from Rx to Tx buffers through memory bus (store and forward architecture). Delays possible due to congestion at the switch
- Each buffer can store multiple packets in FIFO order. Requirements on avionics subsystems to avoid overflow.
- Delay and jitter introduced in message transmission times when waiting for other messages to be transmitted, at end system and at switch
- Multiple switches can be connected
- Redundancy: an AFDX system consists in fact of two independent networks sending the same packets between end systems

AFDX Switch

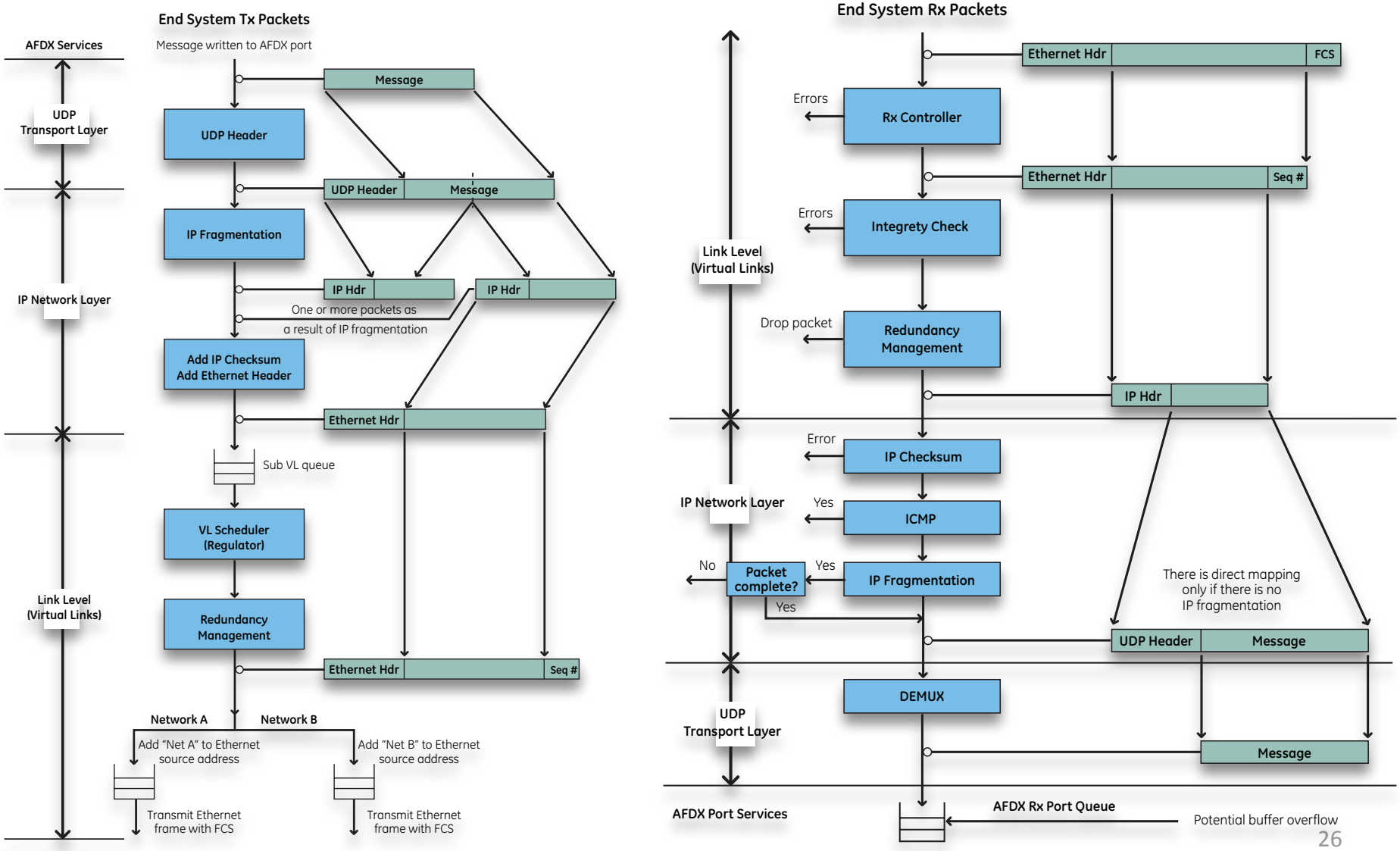


Message Flows

- Packets (AFDX frames, almost identical to Ethernet frames) are sent between End Systems using “Virtual Links” (VLs)
- Total 100 Mbps bandwidth at each end system is shared between VLs
- The bandwidth of each VL is limited: mandatory gaps between messages, max. size of frames
  - bandwidth choice depends on applications connecting to end system via comm. ports
  - bandwidth restrictions enforced by source End Systems, using VL scheduling algorithms.
  - VL scheduler also multiplexes the VL transmission to minimize jitter (req. <= 0.5ms jitter for each VL)



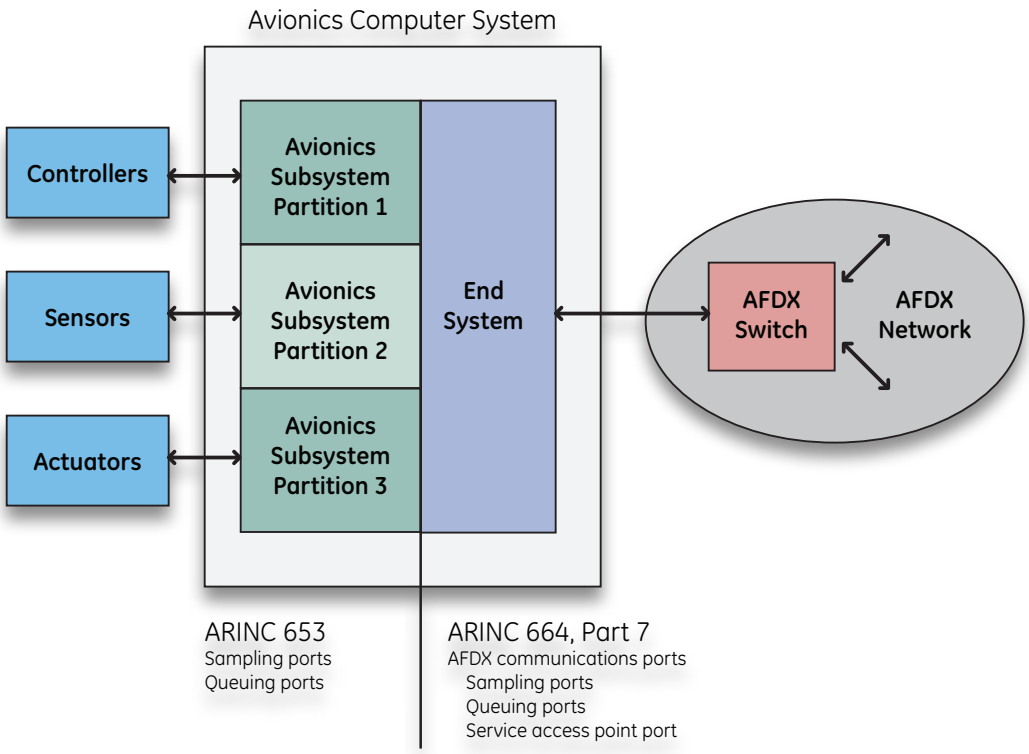
AFDX Tx and Rx Protocol Stacks



Computing Platform Side

Example: ARINC 653

- One computer system partitioned in multiple subsystems
  - restrict address space of each partition
  - limits on amount of CPU time for each partition
- Avionics applications send messages using communication ports
  - communication ports are part of OS API for portable avionics applications described in ARINC 653
  - *Sampling ports* and *Queuing ports*
  - AFDX end system has corresponding sampling and queuing ports, + Service Access Point (SAP) ports for comm. with non-AFDX systems
  - Each sending AFDX comm. ports is associated with a single VL transporting its messages
  - Delay and jitter introduced in message transmission times when waiting for other messages to be transmitted, at end system (VL scheduling) and at switch



ARINC 653/AFDX Communication Ports

- Sampling Ports
  - buffer storage for a single message.
  - a message stays in the buffer until overwritten by new message. Can be read multiple times
  - must provide time-stamped messages. Indication of the freshness of the message in the buffer. Useful for control !
  - Adequate for data sampled for control applications
- Queuing Ports
  - buffer can store a fixed number of message (configuration parameter)
  - reading a message removes it from the queue (FIFO)

Real-Time Scheduling I

- Managing shared computational resources presents challenges similar to managing communication resources
  - single computer system (CPU, ECU = Electronic Control Unit) hosts several applications, critical and non-critical
- Many control applications are implemented on top of a Real-Time Operating System (RTOS), providing resource management services (communication ports, task scheduling, ...)
  - coding, maintenance, integration, changes... unmanageable if done “by hand”, without this service. See CIS 541.
- RT scheduling theory and algorithms developed independently of application
  - flexibility useful to develop complex heterogeneous systems
  - not necessarily compatible with control theoretic assumptions (notion of periodicity is different - execution time vs. interval)
  - control loops generally treated as standard hard real-time tasks

Real-Time Scheduling II

- A task consists of a potentially infinite number of jobs, each requiring access to the CPU to be performed. Can be preemptible or not.
- Each job has a release time, a (bound on) computation or execution time (worst case execution time WCET - need tight timing estimate, maybe difficult) and a deadline. The deadline can be expressed relative to the release time or as an absolute time.
- A task is periodic if the time between the release of successive jobs is a fixed constant, called the period of the task (hence task is time-triggered). It is aperiodic otherwise. Can also distinguish sporadic tasks: aperiodic, with hard deadline (ex: disengage autopilot). Other def. of sporadic: aperiodic task with load bound.
- Objective of the Real-Time scheduler
  - guarantee completion of hard real-time tasks by given deadlines
  - minimize the service time for aperiodic and non hard real-time requests (ex: certain operator request)
- Low-level control tasks are predictable, but scheduler must accommodate other tasks with potentially larger variations to reduce cost of the overall system

Issues for RTOS-Based Digital Controller Implementation I

- Control tasks are not your standard hard real-time task (incompatible abstractions)
  - Do I care if implementation can only give me 1.1ms sampling period instead of 1ms? maybe not, given how I chose the sampling period in the first place... Especially if it is 1.1ms and predictable.
  - On the other hand, do I care if the samples are separated sometimes by 0.1ms, sometimes by 1.9ms?
    - Most likely, yes... Not least because that doesn't match my simple model (recall discretization of systems).
    - could happen with scheduling period = deadline = 1ms in RT scheduling algorithm. Just guarantees execution of the control task once every 1ms.
    - Two very closely separated samples do not give me much new information, actually, so that's possibly just unnecessary load on the resources.
    - Jitter minimization is of some interest in RT scheduling, but that seems to be considered advanced topic. I.e., maybe not available with scheduling policies available on most off-the-shelf RTOS. Want at least to impose minimum gap between executions.

Issues for RTOS-Based Digital Controller Implementation II

- Gap between overall goals of many RTOS, e.g. minimize system response time to external inputs, vs. precise time-triggered sampling assumed by control engineers
- Computation delays are not deterministic and constant
- Good knowledge of RTOS deficiencies can potentially be compensated by the control algorithm
- General purpose RTOS might have to be modified to implement control loops

Real-Time Scheduling Techniques

- Overview of Static, Fixed-Priority, and Dynamic-Priority Scheduling
- See Slides of Prof. Insup Lee, CIS 541

Conclusions

- Complexity of Practical Systems and standards:
  - it is not particularly realistic for the control engineer to claim access to design of protocols
    - ex: AFDX can reuse COTS components and firmware developed for Ethernet protocol
    - cost constraints (e.g. related to development) a big factor
  - In contrast, co-design is a popular research topic in the control literature. you are welcome to follow this path for your project, but should be aware of this. If possible, try to integrate your solution within an existing standard (e.g., can a new protocol be implemented at application layer?)
  - Unfortunately, NECS models in current research tend to vastly oversimplify the problems. We will consider some such models nonetheless, but be aware that there is a lot of room for improvements.

Potentially Useful Tools

- AADL for ARINC 653
  - ENST Tools: OCARINA and POK