15
CAN Performance

18-649 Distributed Embedded Systems

Philip Koopman

October 21, 2015

Carnegie Mellon

Where Are We Now?

- **Where we've been:**
  - CAN – an "event-centric" protocol
- **Where we're going today:**
  - Briefly touch requirements churn (reqd reading from another lecture, but at this point it's relevant to the project – which has a requirements change)
  - Protocol performance, especially CAN
- **Where we're going next:**
  - Scheduling
  - First half of the course – How to make good distributed+embedded systems
  - Second half of the course – How to make them better (dependable; safe; …)

Summary of Requirements Churn (Ch 9)

- **Project 8 adds requirements to make the elevator more realistic**

- **Requirements changes are a fact of life**
  - It is impossible to get 100% of requirements set on Day 1 of project
  - It is, however, a really Bad Idea to just give on requirements because of this

- **The later in the project a requirement changes, the more expensive**
  - "Churn" is when requirements keep changing throughout project
    - Same as other trend; can easily cost 10x-100x more to change late in project
  - It is a relative amount … more churn is worse

- **Usual countermeasures:**
  - Change Control Board: make it painful to insert frivolous changes
  - Requirements Freezes: after a cutoff date you wait until next release
  - Charge for Changes: no change is "free" – it costs money or schedule slip

Preview: CAN Performance

- **A look at workloads and delivery times**
  - Periodic *vs.* aperiodic
  - Best case *vs.* worst case *vs.* expected case

- **A look at CAN protocol performance**
  - Can we predict worst-case delivery time for a CAN message?
    - Perhaps surprisingly, people in the past have said "can't be done" … what they should have said was "not trivial, but can be done"
  - Stay tuned for deadline monotonic scheduling in a later lecture

A Typical Embedded Control Workload

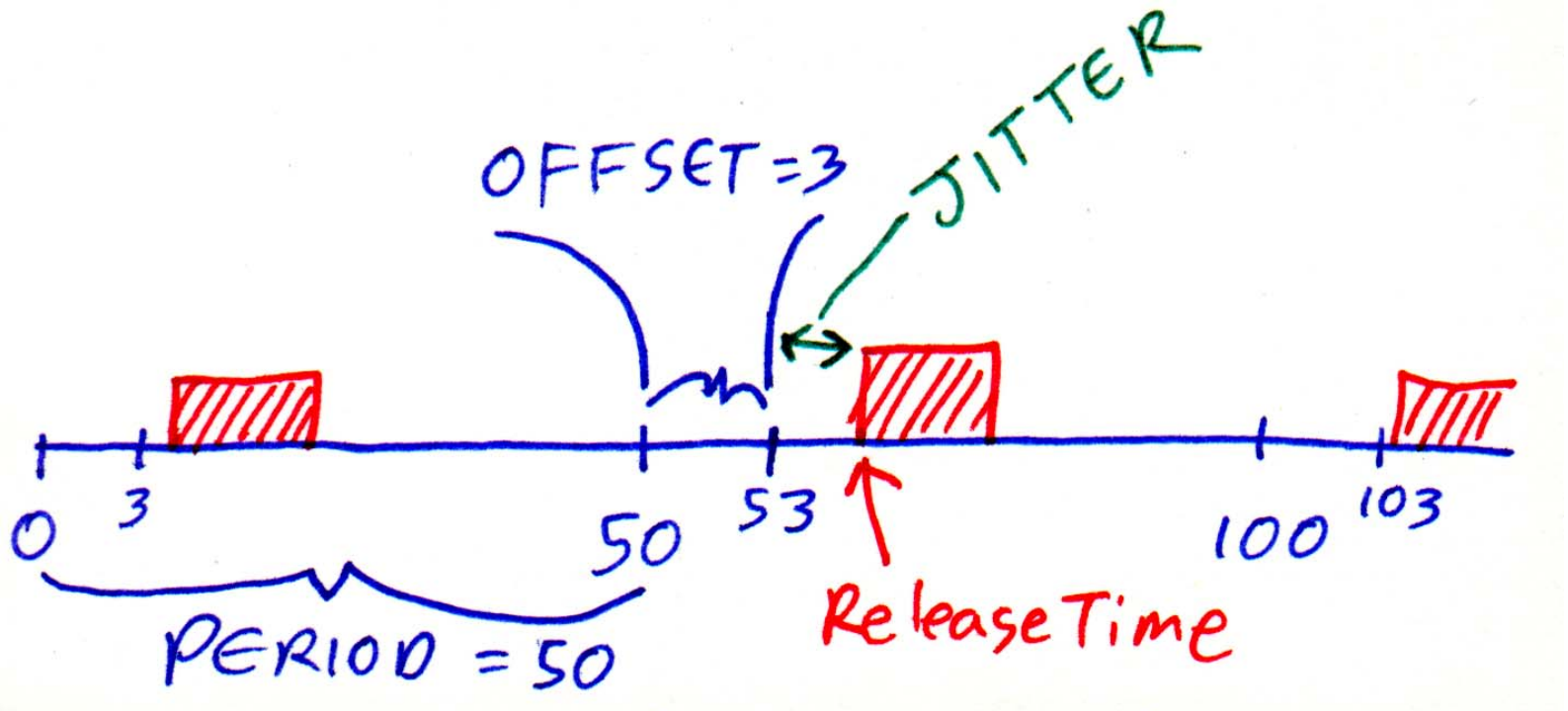
- **"SAE Standard Workload" (subset of 53 messages)** V/C = Vehicle Controller

Signal Number	Signal Description	Size /bits	J /ms	T /ms	Periodic /Sporadic	D /ms	From	To
1	Traction Battery Voltage	8	0.6	100.0	P	100.0	Battery	V/C
2	Traction Battery Current	8	0.7	100.0	P	100.0	Battery	V/C
3	Traction Battery Temp, Average	8	1.0	1000.0	P	1000.0	Battery	V/C
4	Auxiliary Battery Voltage	8	0.8	100.0	P	100.0	Battery	V/C
5	Traction Battery Temp, Max.	8	1.1	1000.0	P	1000.0	Battery	V/C
6	Auxiliary Battery Current	8	0.9	100.0	P	100.0	Battery	V/C
7	Accelerator Position	8	0.1	5.0	P	5.0	Driver	V/C
8	Brake Pressure, Master Cylinder	8	0.1	5.0	P	5.0	Brakes	V/C
9	Brake Pressure, Line	8	0.2	5.0	P	5.0	Brakes	V/C
10	Transaxle Lubrication Pressure	8	0.2	100.0	P	100.0	Trans	V/C
11	Transaction Clutch Line Pressure	8	0.1	5.0	P	5.0	Trans	V/C
12	Vehicle Speed	8	0.4	100.0	P	100.0	Brakes	V/C
13	Traction Battery Ground Fault	1	1.2	1000.0	P	1000.0	Battery	V/C
14	Hi&Lo Contactor Open/Close	4	0.1	50.0	S	5.0	Battery	V/C
15	Key Switch Run	1	0.2	50.0	S	20.0	Driver	V/C
16	Key Switch Start	1	0.3	50.0	S	20.0	Driver	V/C
17	Accelerator Switch	2	0.4	50.0	S	20.0	Driver	V/C
18	Brake Switch	1	0.3	20.0	S	20.0	Brakes	V/C
19	Emergency Brake	1	0.5	50.0	S	20.0	Driver	V/C
20	Shift Lever (PRNDL)	3	0.6	50.0	S	20.0	Driver	V/C

[Tindell94]

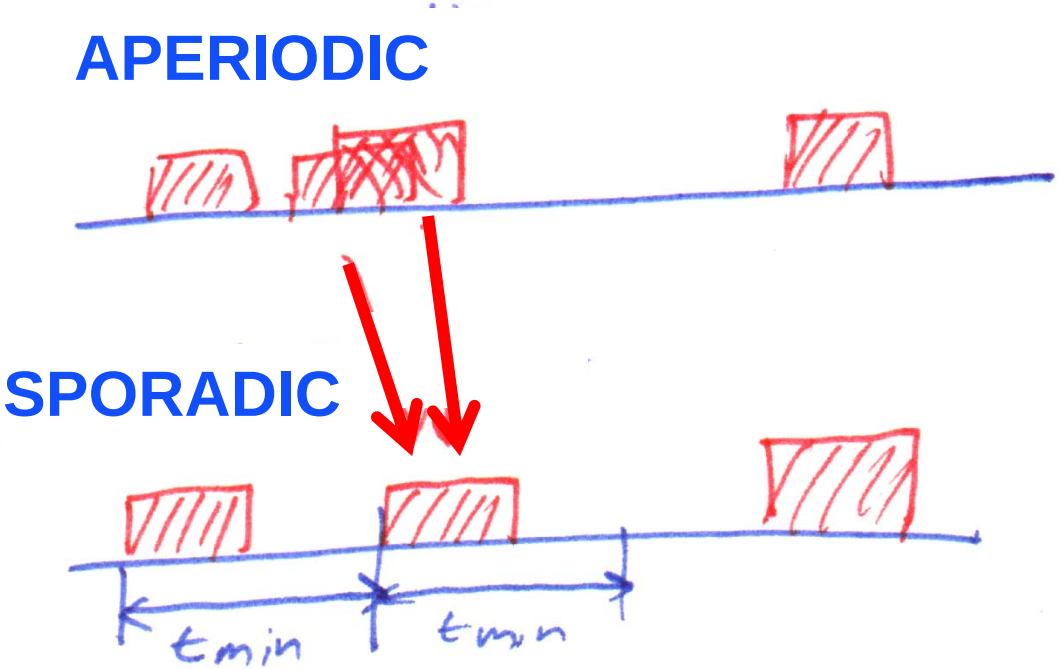
Periodic Messages

- **Time-triggered, often via control loops or rotating machinery**
- **Components to periodic messages**
  - Period (e.g, 50 msec)
  - Offset past period (e.g., 3 msec offset/50 msec period -> 53, 103, 153, 203)
  - Jitter is random "noise" in message release time (*not* oscillator drift)
  - Release time is when message is sent to network queue for transmission
  - Release time$_n$= (n*period) + offset + jitter ; assuming perfect time precision



Sporadic Messages (Aperiodic; ~Exponential)

- **Asynchronous messages**
  - External events
  - Often Poisson processes (exponential inter-arrival times)
- **Sporadic message timing properties**
  - Mean inter-arrival rate (only useful over long time periods)
  - Minimum inter-arrival time with filtering (*often* set equal to deadline)
    - Artificial limit on inter-arrival rate to avoid swamping system ➔ Sporadic
    - May miss arrivals if multiple arrivals occur within the filtering window length



Capacity Measurement

- **Efficiency = amount sent / channel bandwidth**
  - Bit-wise efficiency (data bit encoding; message framing)
  - Payload efficiency (fraction of message that has useful payload)
  - Message efficiency (percent of useful messages)

- **Think of workload demand/network capacity in several ways depending on how long a window you consider:**
  - Instantaneous peak capacity/demand
    - 100% when message is being sent
    - 0% when nothing being sent
  - Worst-case traffic bursts (covered in section on delivery time calculation)
    - What is worst case if a big burst hits? (Someone has to go last)
  - Sustained maximum capacity
    - How many bits can you send with max constant transmitter demand?
    - Most embedded networks avoid collision to get good efficiency under heavy load
    - Usually what you care about is worst case delivery latency at max capacity

Bit-wise Efficiency

- **Intra-message, bit-wise efficiency**
  - Percentage of useful data bits …
    - Data field
    - Portions of header field that can be used to identify the data by an application program (*e.g.*, appropriate parts of CAN message identifier)
  - … compared to total bits transmitted (including overhead bits/dead times)
    - Inter-message gap to permit transmission sources to achieve quiescence
    - Message preamble for receive clock synchronization
    - Control bits
    - Error detection codes
    - Stuff bits
    - Token bits

- **Example: CAN**
  - Say there are 140 bits transmission time for a 64-bit data payload
  - Bit-wise efficiency is (for that message size) is 64 / 140 = **46%**
    - This is pretty good as such things go!

CAN Message Length & Overhead

- **Worst case message length per Ellims *et al.***
  - (Note that older Tindell papers miss subtlety about bit stuffing and incorrectly divides by 5 instead of 4 in equation below)
  - Overhead = 67 bits
    - 29 bit header (slightly different formula below for 11 bit header of course)
    - 15 bit CRC
    - 4 bit length
    - 9 bits start & misc status bits
    - 10 bits end of frame + "intermission" between messages
  - 8 bits/byte of payload (s_m = size of payload in bytes)
  - Worst case of 1 stuff bit for every 4 message bits
    - Why? Because the stuff bit counts as first bit in new stuffing sequence!
    - Only 54 of the overhead bits are stuffed; Intermission and some others not stuffed

$$\#bits = \left(\left\lfloor \frac{54 + 8s_m}{4} \right\rfloor + 67 + 8s_m \right)$$

[Ellims02]

Message Use Efficiency

- **How many messages are actually used?**
  - In any token/polled system, percentage of data-bearing messages vs. empty token passes
  - Might make assumption of uniform message length; might be only somewhat accurate
- **Master/Slave system**
  - Efficiency might be 50% (half of messages are polls; half have data)
- **Token passing system**
  - Efficiency varies depending on system load (more efficient at high loads)
  - Efficiency critically depends on combining token with useful messages
- **Collision-based systems**
  - Efficiency depends on collisions
  - Efficiency *reduces* with busy system (this is undesirable)
- **CAN:**
  - 100% of messages are useful at protocol level

Tricks To Improve Efficiency

- **Combine messages into large messages**
  - Several different data fields put into a single message
    - Be careful – only works for messages sent from same transmitter
    - Can obscure event triggers by combining two events into one message

- **Plan message spacing to minimize arbitration overhead**
  - If there is startup cost to achieve active network, intentionally clump messages (keep offset low, but expect to have long latency messages)
  - If synchronized messages collide and cost performance, intentionally skew message release times (add jitter)

  - Time-triggered approaches using TDMA *can* be 100% message efficient
    - But, sometimes the messages are sending redundant data
    - And, usually requires precise timekeeping

Average Demand

- **Average Demand is based on mean periods**
  - Periodic messages – at stated period
  - Aperiodic messages – at stated mean period
  - Assume time period is long enough that message clumping doesn't matter

  - For example workload over 30 time units:
    (*note:* 30 is Least Common Multiple of periods)

Message Name	Type	Mean Period	# in 30 units
A	Periodic	1	30
B	Sporadic	5	6
C	Periodic	5	6
D	Periodic	10	3
E	Periodic	5	6
F	Sporadic	10	3
G	Periodic	30	1

Total = 55 messages / 30 time units

Peak Demand

- **For critical systems, you have to plan on the worst case!**
- **Peak demand is based on mean periods + max arrival rates**
  - Periodic messages – at stated period
  - Aperiodic messages – at minimum arrival intervals (peak period)
    - ***Usually you assume minimum intervals = deadline***
  - Assume time period is long enough that message clumping doesn't matter
  - For example sustained peak workload over 30 time units:

Message Name	Type	Mean Period	Peak Period	# in 30 units
A	Periodic	1		30
B	Sporadic	5	2	15
C	Periodic	5		6
D	Periodic	10		3
E	Periodic	5		6
F	Sporadic	10	10	3
G	Periodic	30		1

Total = 64 messages / 30 time units

Abbreviated "Standard" Automotive Workload

Signal Number	*Signal Description*	*Size /bits*	*J /ms*	*Period /ms*	*Periodic /Sporadic*	*Deadline /ms*	*From*	*To*
1	T_Batt_V	8	0.6	100	P	100	Battery	V/C
2	T_Batt_C	8	0.7	100	P	100	Battery	V/C
3	T_Batt_Tave	8	1.0	1000	P	1000	Battery	V/C
4	A_Batt_V	8	0.8	100	P	100	Battery	V/C
5	T_Batt_Tmax	8	1.1	1000	P	1000	Battery	V/C
6	A_Batt_C	8	0.9	100	P	100	Battery	V/C
7	Accel_Posn	8	0.1	5	P	5	Driver	V/C
8	Brake_Master	8	0.1	5	P	5	Brakes	V/C
9	Brake_Line	8	0.2	5	P	5	Brakes	V/C
10	Trans_Lube	8	0.2	100	P	100	Trans	V/C
11	Trans_Clutch	8	0.1	5	P	5	Trans	V/C
12	Speed	8	0.4	100	P	100	Brakes	V/C
13	T_Batt_GF	1	1.2	1000	P	1000	Battery	V/C
14	Contactor	4	0.1	50	S	5	Battery	V/C
15	Key_Run	1	0.2	50	S	20	Driver	V/C
16	Key_Start	1	0.3	50	S	20	Driver	V/C
17	Accel_Switch	2	0.4	50	S	20	Driver	V/C
18	Brake_Switch	1	0.3	20	S	20	Brakes	V/C
19	Emer_Brake	1	0.5	50	S	20	Driver	V/C
20	Shift_Lever	3	0.6	50	S	20	Driver	V/C

Note on Deadline Monotonic Scheduling

- **If you've never heard about this, be sure to read about it before the scheduling lecture!**
  - Rate Monotonic and Deadline Monotonic are almost the same idea
  - Generally talks about CPU scheduling
  - But, with slight adjustment works on network message schedules too!

- **Generalized way to meet real time deadlines with prioritized tasks:**
  - Sort messages by priority order
  - Shortest period gets highest priority
  - If rates are harmonic (all rates multiple of other rates), can use 100% of resource
  - More about this in the real time scheduling lecture

Sort For Deadline Monotonic Scheduling

Signal Number	Signal Description	Size /bits	J /ms	Period /ms	Periodic /Sporadic	Deadline /ms	From
7	Accel_Posn	8	0.1	5	P	5	Driver
8	Brake_Master	8	0.1	5	P	5	Brakes
9	Brake_Line	8	0.2	5	P	5	Brakes
11	Trans_Clutch	8	0.1	5	P	5	Trans
14	Contactor	4	0.1	50	S	5	Battery
15	Key_Run	1	0.2	50	S	20	Driver
16	Key_Start	1	0.3	50	S	20	Driver
17	Accel_Switch	2	0.4	50	S	20	Driver
18	Brake_Switch	1	0.3	20	S	20	Brakes
19	Emer_Brake	1	0.5	50	S	20	Driver
20	Shift_Lever	3	0.6	50	S	20	Driver
1	T_Batt_V	8	0.6	100	P	100	Battery
2	T_Batt_C	8	0.7	100	P	100	Battery
4	A_Batt_V	8	0.8	100	P	100	Battery
6	A_Batt_C	8	0.9	100	P	100	Battery
10	Trans_Lube	8	0.2	100	P	100	Trans
12	Speed	8	0.4	100	P	100	Brakes
3	T_Batt_Tave	8	1.0	1000	P	1000	Battery
5	T_Batt_Tmax	8	1.1	1000	P	1000	Battery
13	T_Batt_GF	1	1.2	1000	P	1000	Battery

Workload Bandwidth Consumption

- **Total Bandwidth = 122,670 bits/sec** (worst case)

Signal Number	*Signal Description*	*Size /bits*	*J /ms*	*Period /ms*	*Periodic /Sporadic*	*Deadline /ms*	*From*	*bits/ message*	*bits/ second*
7	Accel_Posn	8	0.1	5	P	5	Driver	90	18000
8	Brake_Master	8	0.1	5	P	5	Brakes	90	18000
9	Brake_Line	8	0.2	5	P	5	Brakes	90	18000
11	Trans_Clutch	8	0.1	5	P	5	Trans	90	18000
14	Contactor	4	0.1	50	S	5	Battery	90	18000
15	Key_Run	1	0.2	50	S	20	Driver	90	4500
16	Key_Start	1	0.3	50	S	20	Driver	90	4500
17	Accel_Switch	2	0.4	50	S	20	Driver	90	4500
18	Brake_Switch	1	0.3	20	S	20	Brakes	90	4500
19	Emer_Brake	1	0.5	50	S	20	Driver	90	4500
20	Shift_Lever	3	0.6	50	S	20	Driver	90	4500
1	T_Batt_V	8	0.6	100	P	100	Battery	90	900
2	T_Batt_C	8	0.7	100	P	100	Battery	90	900
4	A_Batt_V	8	0.8	100	P	100	Battery	90	900
6	A_Batt_C	8	0.9	100	P	100	Battery	90	900
10	Trans_Lube	8	0.2	100	P	100	Trans	90	900
12	Speed	8	0.4	100	P	100	Brakes	90	900
3	T_Batt_Tave	8	1.0	1000	P	1000	Battery	90	90
5	T_Batt_Tmax	8	1.1	1000	P	1000	Battery	90	90
13	T_Batt_GF	1	1.2	1000	P	1000	Battery	90	90

Schedulability (Trivial Bound Version)

- **Look at shortest period and see if everything fits there**

- **Shortest period = 5 msec**
  - 20 messages total, each at 90 bits = 20*90= 1800 bits if all messages are released simultaneously
  - 1800 bits / 5 msec = 360,000 bits/sec
  - Therefore, system is trivially schedulable above 360,000 bits/sec (i.e., all 20 messages would be schedulable if they were all sent at 5 msec periods)

Signal Description	*Size /bits*	*Deadline /ms*	*From*	*bits/ message*	*bits/ second*
Accel_Posn	8	5	Driver	90	18000
Brake_Master	8	5	Brakes	90	18000
Brake_Line	8	5	Brakes	90	18000
Trans_Clutch	8	5	Trans	90	18000
Contactor	4	5	Battery	90	18000
Key_Run	1	20 -> 5	Driver	90	18000
Key_Start	1	20 -> 5	Driver	90	18000
Accel_Switch	2	20 -> 5	Driver	90	18000
Brake_Switch	1	20 -> 5	Brakes	90	18000
Emer_Brake	1	20 -> 5	Driver	90	18000
Shift_Lever	3	20 -> 5	Driver	90	18000
T_Batt_V	8	100 -> 5	Battery	90	18000
T_Batt_C	8	100 -> 5	Battery	90	18000
A_Batt_V	8	100 -> 5	Battery	90	18000
A_Batt_C	8	100 -> 5	Battery	90	18000
Trans_Lube	8	100 -> 5	Trans	90	18000
Speed	8	100 -> 5	Brakes	90	18000
T_Batt_Tave	8	1000 -> 5	Battery	90	18000
T_Batt_Tmax	8	1000 -> 5	Battery	90	18000
T_Batt_GF	1	1000 -> 5	Battery	90	18000
			Total bits/sec =		360,000

Schedulability (Deadline Monotonic Version)

- **Assign priorities based on shortest deadlines; Network Load < ~100%**
  - Result is schedulable at ~125 Kbps (note that deadlines are harmonic)

Signal Number	*Priority*	*Signal Description*	*Size /bits*	*J /ms*	*Period /ms*	*Periodic /Sporadic*	*Deadline /ms*	*From*	*bits/ message*	*bits/ second*
7	***1***	Accel_Posn	8	0.1	5	P	5	Driver	90	18000
8	***2***	Brake_Master	8	0.1	5	P	5	Brakes	90	18000
9	***3***	Brake_Line	8	0.2	5	P	5	Brakes	90	18000
11	***4***	Trans_Clutch	8	0.1	5	P	5	Trans	90	18000
14	***5***	Contactor	4	0.1	50	S	5	Battery	90	18000
15	***6***	Key_Run	1	0.2	50	S	20	Driver	90	4500
16	***7***	Key_Start	1	0.3	50	S	20	Driver	90	4500
17	***8***	Accel_Switch	2	0.4	50	S	20	Driver	90	4500
18	***9***	Brake_Switch	1	0.3	20	S	20	Brakes	90	4500
19	***10***	Emer_Brake	1	0.5	50	S	20	Driver	90	4500
20	***11***	Shift_Lever	3	0.6	50	S	20	Driver	90	4500
1	***12***	T_Batt_V	8	0.6	100	P	100	Battery	90	900
2	***13***	T_Batt_C	8	0.7	100	P	100	Battery	90	900
4	***14***	A_Batt_V	8	0.8	100	P	100	Battery	90	900
6	***15***	A_Batt_C	8	0.9	100	P	100	Battery	90	900
10	***16***	Trans_Lube	8	0.2	100	P	100	Trans	90	900
12	***17***	Speed	8	0.4	100	P	100	Brakes	90	900
3	***18***	T_Batt_Tave	8	1.0	1000	P	1000	Battery	90	90
5	***19***	T_Batt_Tmax	8	1.1	1000	P	1000	Battery	90	90
13	***20***	T_Batt_GF	1	1.2	1000	P	1000	Battery	90	90
								Total bits/sec =		122,670

Can We Do Better?

- **Look for messages to combine**
  - Even 1-bit payloads consume a whole message
  - Look for messages with *same period* and *same source*

Signal Number	*Signal Description*	*Size /bits*	*J /ms*	*Period /ms*	*Periodic /Sporadic*	*Deadline /ms*	*From*	*bits/ message*	*bits/ second*	
14	Contactor	4	0.1	50	S	5	Battery	90	18000	
8	Brake_Master	8	0.1	5	P	5	Brakes	90	18000	**Brake_msg**
9	Brake_Line	8	0.2	5	P	5	Brakes	90	18000	
7	Accel_Posn	8	0.1	5	P	5	Driver	90	18000	
11	Trans_Clutch	8	0.1	5	P	5	Trans	90	18000	
18	Brake_Switch	1	0.3	20	S	20	Brakes	90	4500	
15	Key_Run	1	0.2	50	S	20	Driver	90	4500	
16	Key_Start	1	0.3	50	S	20	Driver	90	4500	
17	Accel_Switch	2	0.4	50	S	20	Driver	90	4500	**Driver_msg**
19	Emer_Brake	1	0.5	50	S	20	Driver	90	4500	
20	Shift_Lever	3	0.6	50	S	20	Driver	90	4500	
1	T_Batt_V	8	0.6	100	P	100	Battery	90	900	
2	T_Batt_C	8	0.7	100	P	100	Battery	90	900	**Batt_msg1**
4	A_Batt_V	8	0.8	100	P	100	Battery	90	900	
6	A_Batt_C	8	0.9	100	P	100	Battery	90	900	
12	Speed	8	0.4	100	P	100	Brakes	90	900	
10	Trans_Lube	8	0.2	100	P	100	Trans	90	900	
3	T_Batt_Tave	8	1.0	1000	P	1000	Battery	90	90	
5	T_Batt_Tmax	8	1.1	1000	P	1000	Battery	90	90	**Batt_msg2**
13	T_Batt_GF	1	1.2	1000	P	1000	Battery	90	90	
							Total bits/sec =		122,670	

Workload With Combined Messages

- **Deadline monotonic schedulable at >86,110 bits/sec (36 Kbps savings)**

Priority	*Signal Number*	*Signal Description*	*Size /bits*	*J /ms*	*Period /ms*	*Periodic /Sporadic*	*Deadline /ms*	*From*	*bits/ message*	*bits/ second*
1	14	Contactor	4	0.1	50	S	5	Battery	90	18000
2	**8+9**	**Brake_msg**	16	0.1	5	P	5	Brakes	**100**	20000
3	7	Accel_Posn	8	0.1	5	P	5	Driver	90	18000
4	11	Trans_Clutch	8	0.1	5	P	5	Trans	90	18000
5	18	Brake_Switch	1	0.3	20	S	20	Brakes	90	4500
6	**15-17,19-20**	**Driver_msg**	8	0.2	50	S	20	Driver	**90**	4500
7	**1,2,4,6**	**Batt_msg1**	32	0.6	100	P	100	Battery	**120**	1200
8	12	Speed	8	0.4	100	P	100	Brakes	90	900
9	10	Trans_Lube	8	0.2	100	P	100	Trans	90	900
10	**3,5,13**	**Batt_msg2**	17	1.0	1000	P	1000	Battery	**110**	110
								Total bits/sec =		86,110

- **But, not always a free lunch**
  - What if design changes to have a different message source for one message?
  - What if design needs to change deadline of one of a combined message?
  - What if portions of Driver_msg have “event” semantics and aren’t always sent?

Message Latency

- **Networks are an inherently serial medium**
  - In the worst case, *some* message is going to go last

- **Latency for our purposes starts when you queue a message and ends when message is received:**
  - Minimum theoretical latency is therefore the length of the message itself
  - End-to-end latency might also include:
    - Sensor/OS/application/OS/NIC delay at transmitter
    - NIC/OS/application/OS/actuator delay at receiver
  - Latency is measured until *after the last bit* of the message is received
    - Message isn't received until everything included error codes are received
    - PLUS need to add processing delay to check error code, etc.
  - If a message is enqueued just 1 psec after node started transmitting or gave up slot in arbitration, message has to wait for next opportunity
    - Message can't transmit unless it was enqueued ***before*** current arbitration round begins
    - Even if you could theoretically slip it in later – that is not how networks are built

Round-Robin Message Latency

- **Best case is message transmits immediately (gets lucky or network idle)**

- **Worst case is message has to wait for its turn**
  - Message might have just barely missed current turn
  - Wait for current message to complete
  - Wait for all other nodes in a round to transmit
  - Transmit desired message
  - Possibly, wait for *k+1* rounds if there are *k* messages already enqueued at the transmitter node

Prioritized Message Latency

- **Best case – message transmits immediately**

- **Prioritized messages worst case:**
  - Currently transmitting message completes
    - You *do not* pre-empt a message once it starts transmitting
  - All higher-priority messages complete
  - Potentially, all other messages of same priority complete
  - Desired message completes
  - NOTE: node number of origin doesn't matter for globally prioritized messages
    - (assuming prioritization is by message ID # rather than node number)

- **Message latency is analogous to prioritized interrupts**
  - 18-348 material – for this course you just need to apply this to CAN network messages, but the ideas are very similar.

Prioritized CPU Interrupts Are Similar Idea

Higher

Priority

Lower

Vector Address	Interrupt Source	CCR Mask	Local Enable	HPRIO Value to Elevate
0xFFFE, 0xFFFF	External reset, power on reset, or low voltage reset (see CRG flags register to determine reset source)	None	None	—
0xFFFC, 0xFFFD	Clock monitor fail reset	None	COPCTL (CME, FCME)	—
0xFFFA, 0xFFFB	COP failure reset	None	COP rate select	—
0xFFF8, 0xFFF9	Unimplemented instruction trap	None	None	—
0xFFF6, 0xFFF7	SWI	None	None	—
0xFFF4, 0xFFF5	XIRQ	X-Bit	None	—
0xFFF2, 0xFFF3	IRQ	I bit	INTCR (IRQEN)	0x00F2
0xFFF0, 0xFFF1	Real time Interrupt	I bit	CRGINT (RTIE)	0x00F0
0xFFEE, 0xFFEF	Standard timer channel 0	I bit	TIE (C0I)	0x00EE
0xFFEC, 0xFFED	Standard timer channel 1	I bit	TIE (C1I)	0x00EC
0xFFEA, 0xFFEB	Standard timer channel 2	I bit	TIE (C2I)	0x00EA
0xFFE8, 0xFFE9	Standard timer channel 3	I bit	TIE (C3I)	0x00E8
0xFFE6, 0xFFE7	Standard timer channel 4	I bit	TIE (C4I)	0x00E6
0xFFE4, 0xFFE5	Standard timer channel 5	I bit	TIE (C5I)	0x00E4
0xFFE2, 0xFFE3	Standard timer channel 6	I bit	TIE (C6I)	0x00E2
0xFFE0, 0xFFE1	Standard timer channel 7	I bit	TIE (C7I)	0x00E0
0xFFDE, 0xFFDF	Standard timer overflow	I bit	TMSK2 (TOI)	0x00DE
0xFFDC, 0xFFDD	Pulse accumulator A overflow	I bit	PACTL (PAOVI)	0x00DC

Latency For Prioritized Interrupts

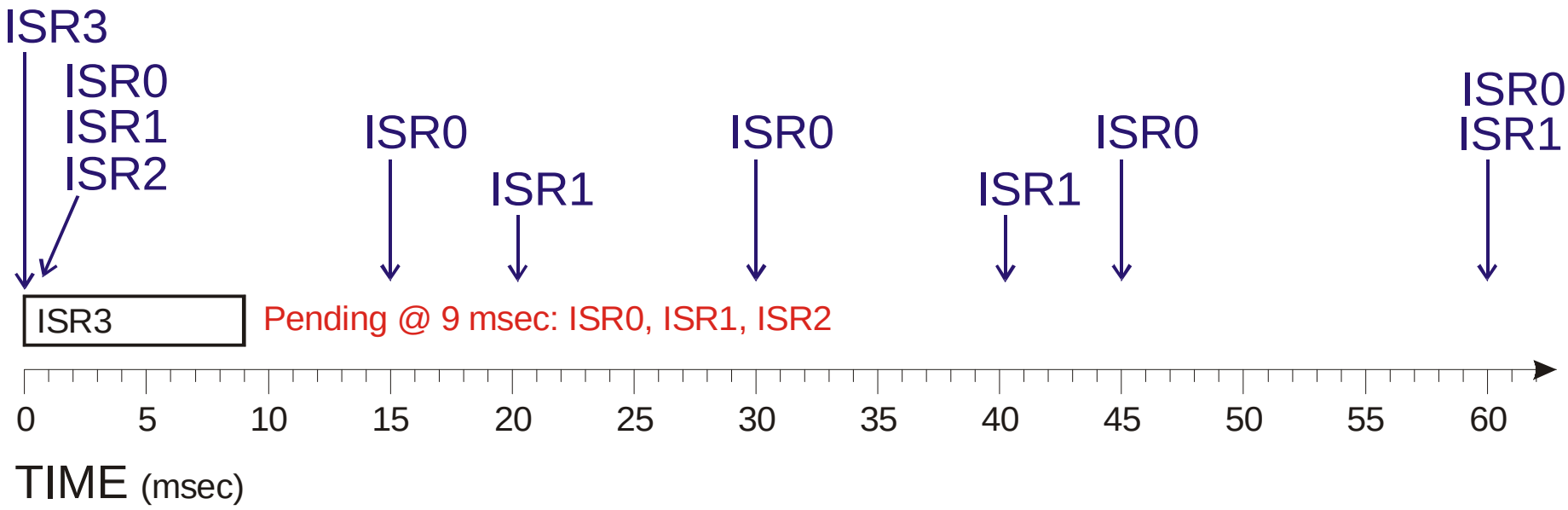
- **Have to wait for other interrupts to execute**
  - One might already be executing with lower priority (have to wait)
    - Or, interrupts might be masked for some other reason ("blocking")
  - All interrupts at higher priority might execute one or more times
  - Worst case – have to assume that every possible higher priority interrupt is queued AND longest possible blocking time (lower priority interrupt)

- **Example, (same as previous situation):**
  - ISR1 takes 1 msec; repeats at most every 10 msec
  - ISR2 takes 2 msec; repeats at most every 20 msec
  - ISR3 takes 3 msec; repeats at most every 30 msec

  - For ISR2, latency is:
    - ISR3 might just have started – 3 msec
    - ISR1 might be queued already – 1 msec
    - ISR2 will run after 3 + 1 = 4 msec
      - » This is less than 10 msec total (period of ISR1), so ISR1 doesn't run a second time

Example – ISR Worst Case Latency

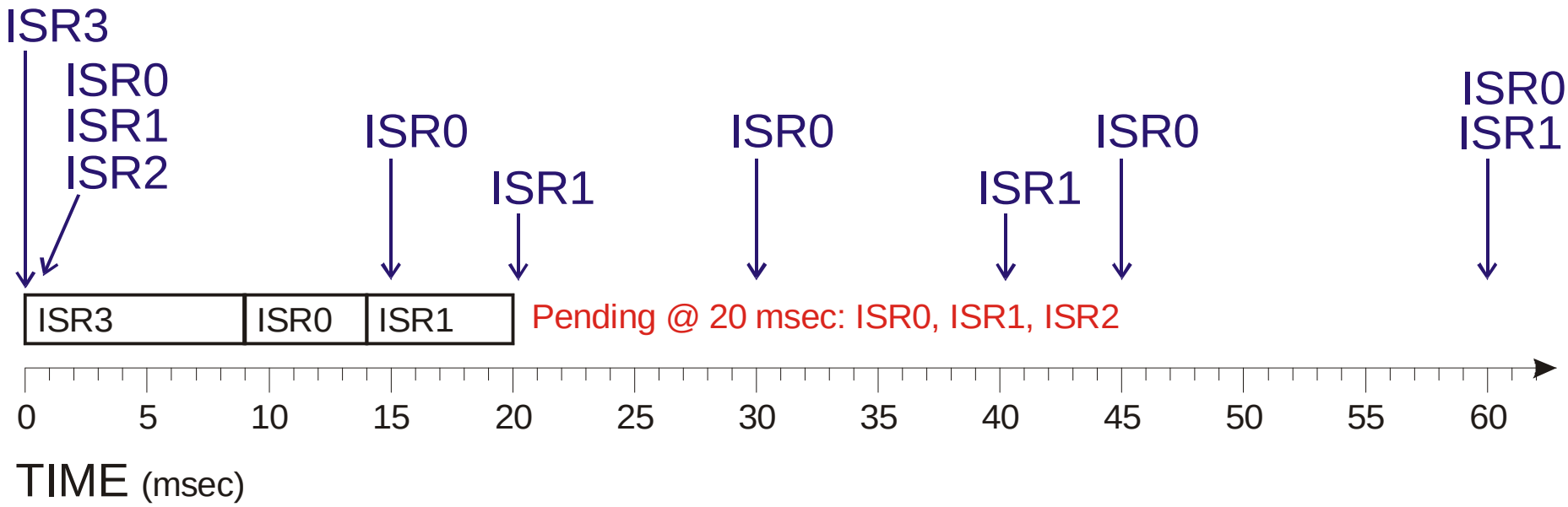
- **Assume following task set (ISR0 highest priority):**
  - ISR0 takes 5 msec and occurs at most once every 15 msec
  - ISR1 takes 6 msec and occurs at most once every 20 msec
  - **ISR2** takes 7 msec and occurs at most once every 100 msec
  - ISR3 takes 9 msec and occurs at most once every 250 msec
  - ISR4 takes 3 msec and occurs at most once every 600 msec



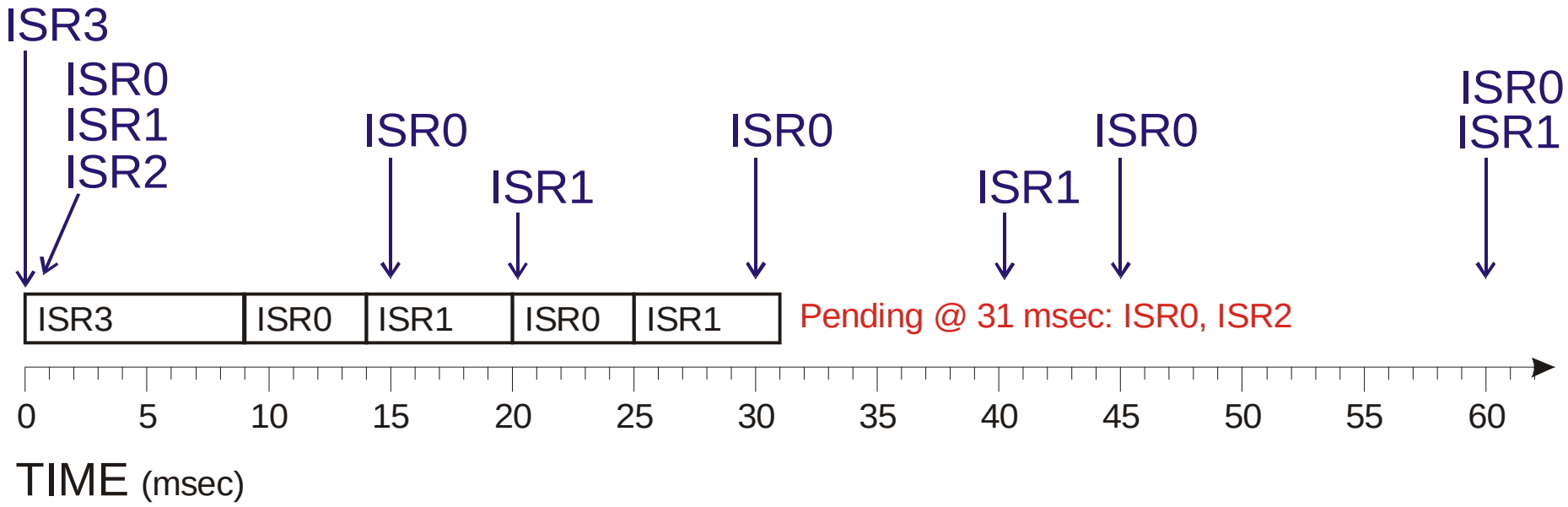
Will ISR2 Execute Within 50 msec?

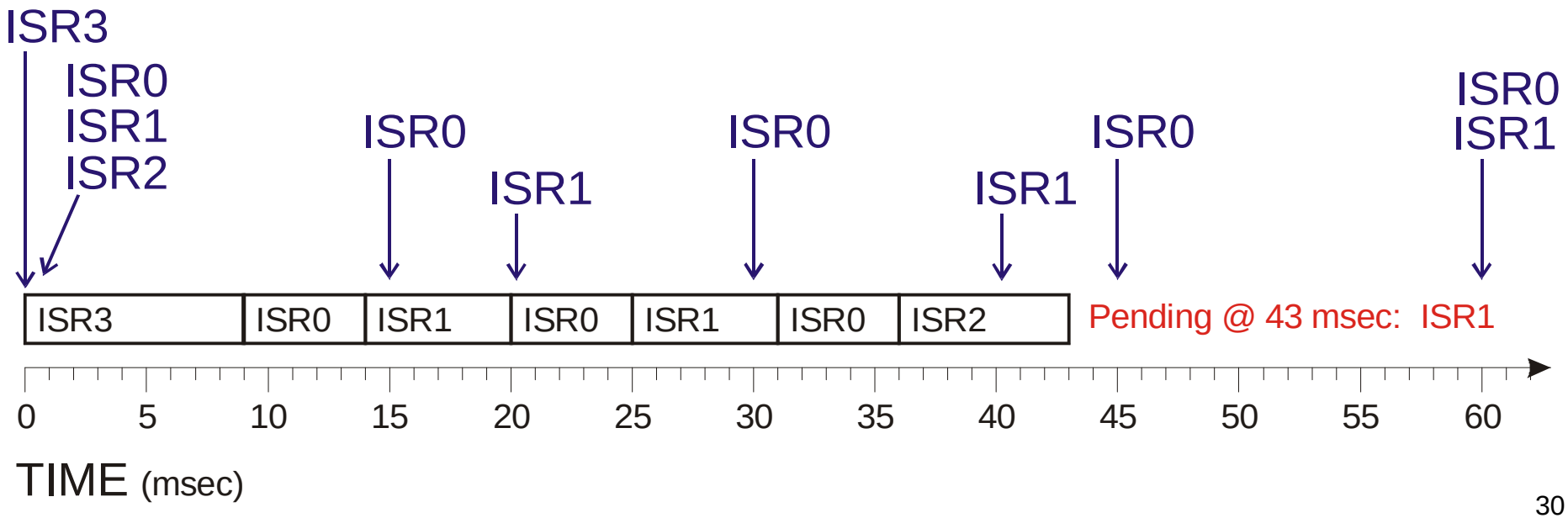
- **Worst Case is ISR3 runs just before ISR2 can start**
  - Why this one? – has longest execution time of everything lower than ISR2

- **Then ISR0 & ISR1 go because they are higher priority**
  - But wait, they retrigger by 20 msec – so they are pending *again*



ISR0 & ISR1 Retrigger, then ISR2 goes





ISR Latency – The Math

- **In general, higher priority interrupts might run multiple times!**
  - Assume N different interrupts sorted by priority (0 is highest; N-1 is lowest)
  - Want latency of interrupt *m*

$$ilatency_0 = 0$$

$$ilatency_{i+1} = \max_{j>m}\left(ISRtime_j\right) + \sum_{\forall ISRs_{j<m}} \left\lfloor \frac{ilatency_i}{ISRperiod_j} + 1 \right\rfloor ISRtime_j$$

  - What it's saying is true for anything with prioritization plus initial blocking time:
    1. You have to wait for *one worst-case* task at same or lower priority to complete
    2. You always have to wait for all tasks with higher priority, sometimes repeated

Example Response Time Calculation

- **What's the Response Time for task 2?**
  - Note: N=4 (tasks 0..3)
  - Have to wait for task 3 to finish
    - (longest execution time)
  - Have to wait for two execution of task 0
  - Have to wait for one execution of task 1

Task# *i*	Period (P_i)	Execution Time (C_i)
0	**8**	**1**
1	**12**	**2**
2	**20**	**3**
3	**25**	**6**

$$R_{2,0} = \max_{2<j<4}(C_j) = C_3 = 6$$

$$R_{2,1} = R_{2,0} + \sum_{m=0}^{m=1}\left(\left\lceil \frac{R_{i,0}}{P_m} + 1 \right\rceil C_m\right) = 6 + \left(\left\lceil \frac{6}{8} + 1 \right\rceil 1\right) + \left(\left\lceil \frac{6}{12} + 1 \right\rceil 2\right) = 6 + 1 + 2 = 9$$

$$R_{2,2} = R_{2,0} + \sum_{m=0}^{m=1}\left(\left\lceil \frac{R_{i,1}}{P_m} + 1 \right\rceil C_m\right) = 6 + \left(\left\lceil \frac{9}{8} + 1 \right\rceil 1\right) + \left(\left\lceil \frac{9}{12} + 1 \right\rceil 2\right) = 6 + 2 + 2 = 10$$

$$R_{2,\infty} = 10$$

Blocking Delay

- **CAN is not a purely preemptive system**
  - Messages queue while waiting for previous message to transmit
  - This aspect of scheduling is *non-preemptive – just like ISRs on most CPUs*

- **Blocking time while waiting for previous message:** ([Ellims02])

$$B_m = \max_{\forall k \in lp(m)} (C_k)$$

  - ~"Blocking time is longest possible message that could have just started transmission"

  - For combined message example, longest time is 120 bits for **Batt_msg1**
    - Do we need to worry about this for schedulability?
    - In general, yes, especially if 120 bits is non-trivial with respect to deadlines

  - Next, let's look at delivery time for individual messages

Example Worst-Case Timing

- **Bound results:**
  - Look at maximum # higher priority messages within deadline
    - Gives pessimistic analysis of worst case pre-emption
    - (Don't forget to add blocking message)

						Bits sent within n msec			
Priority	***Signal Description***	***J /ms***	***Deadline /ms***	***From***	***bits/ message***	**5**	**20**	**100**	**1000**
1	Contactor	0.1	5	Battery	90	90	360	1800	18000
2	Brake_msg	0.1	5	Brakes	100	100	400	2000	20000
3	Accel_Posn	0.1	5	Driver	90	90	360	1800	18000
4	Trans_Clutch	0.1	5	Trans	90	90	360	1800	18000
5	Brake_Switch	0.3	20	Brakes	90		90	450	4500
6	**Driver_msg**	**0.2**	**20**	**Driver**	**90**		**90**	450	4500
7	Batt_msg1	0.6	100	Battery	**120**	**120**	**120**	120	1200
8	Speed	0.4	100	Brakes	90			90	900
9	Trans_Lube	0.2	100	Trans	90			90	900
10	Batt_msg2	1.0	1000	Battery	110		110	110	110

- Example: **Driver_msg** has to wait for:
  - Batt_msg1 as potential blocking message (120 bits)
  - Four copies of each 5 msec message + 1 copy of Brake_switch
  - For this example, 120+360+400+360+360+90+90 bits (assuming it meets deadline)

Why Bother With All This Stuff?

- **Deadline monotonic scheduling (later lecture) is a general-purpose tool**
  - BUT – you can beat deadline monotonic in some situations by using exact schedulability approaches
- **Ellim's equation gives a more exact result**
  - Can do better if you account for offset, especially on messages coming from same processor
  - Can do better if you account for multiple messages being at same rate even though they have different deadlines
  - Does NOT include retransmissions/lost message effects
    - These make things worse; need to allocate a budget and hope you don't exceed it
- **Now you can see how to get static scheduling on CAN**
  - Launch all copies of messages at exactly zero offset + zero jitter
  - Messages empty out of transmission queues according to priority
  - Gives static schedule with just a single, periodic "clock" message to trigger all the message releases

Other Capacity Issues

- **Message acknowledgements**
  - Broadcast messages may generate a message-ack flurry on some systems
  - CAN uses single ACK & error frame NACK for all receivers
    - Helps deal with localized noise sources
  - Other protocols may require distinct Ack from all receivers
    - e.g., 1 message + 8 acks for a single message in the workload

- **Message retries**
  - Errors may require retransmission; leave slack space for that

- **"Headroom"**
  - Good system architects include 4x or 5x "headroom" into new systems
    - Can it handle the same traffic with 5x faster frequency?
    - Is that traffic from the same number of nodes or from 5x more nodes?
    - This accounts for 10-20 years of system evolution

Receiver Over-Run

- **Slow receivers can be over-run with fast messages**
  - Assume interface hardware can catch messages…
    but that slow CPU must remove them from receive queue

- **Possible approaches**
  - Throttle all message transmissions with large inter-message gap (Lonworks)
    - CAN has a 7 bit “intermission” for this purpose, and “overload frame” if node needs a little more time
  - Require receiver to indicate ready-to-receive before transmitting
    - Or, retry on receive-buffer overflow
  - Send only *Q* message types to slow nodes, where *Q* is receive queue depth
    - Combine messages into a single message
    - Use separate “mailboxes”, one for each of the *Q* types (CAN)
  - Deliberately schedule messages so that receivers are never over-run

Tricks To Improve Latency

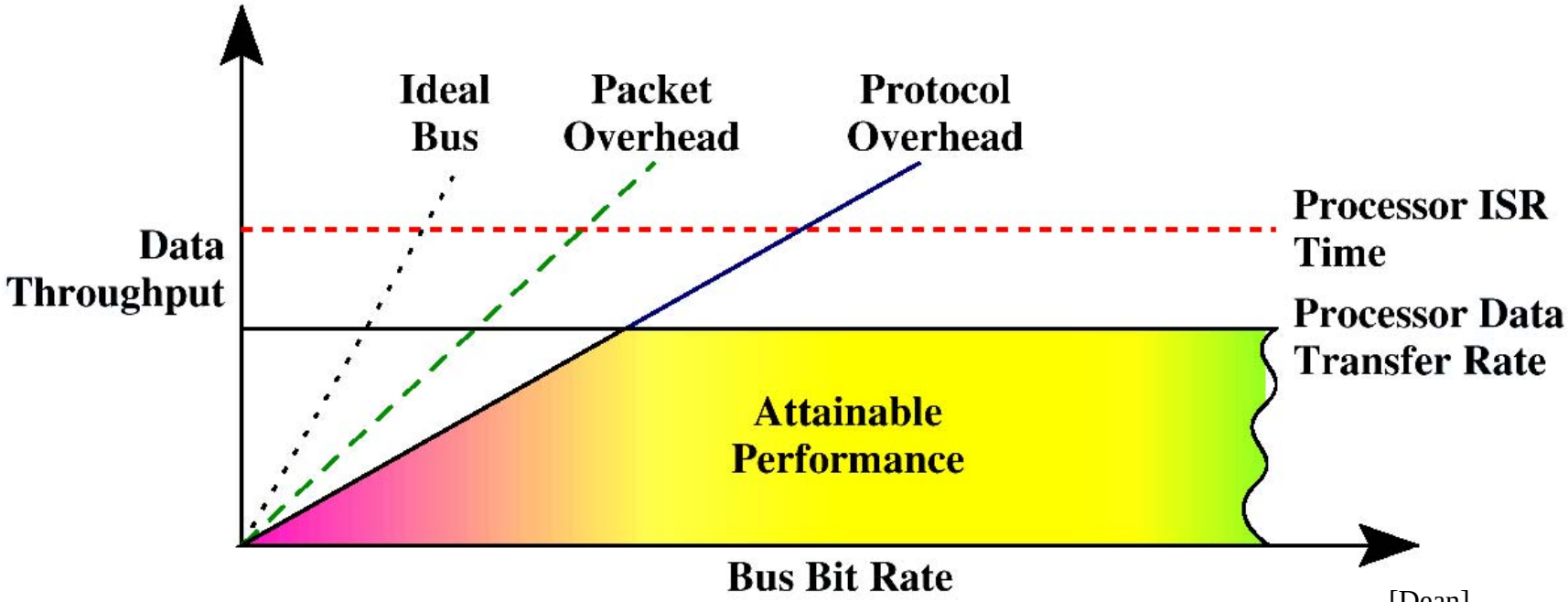
- **Schedule message offsets to avoid conflicts/waits**
  - Ultimate is TTP, which pre-schedules messages for zero conflicts
    - That makes it a TDMA protocol
  - If tasks produce messages just in time, latency is simply transmission time

- **Schedule tasks in order of output message priority**
  - No point scheduling tasks that produce low priority messages early in a cycle … they'll just have to wait to transmit anyway
  - Might even intentionally add some delay to spread tasks out in cycle

- **The Fast CPU = Poor Network Performance paradox (!)**
  - A faster CPU can increase message latency
  - Enqueuing low-priority messages quickly simply gives them longer to rot in the output queue

Limits to Performance

- **Payload within message**
  - Bits for header, error detection, etc.
- **Message encoded into bits**
  - Bandwidth used for self-clocking, stuff bits, message framing
- **Arbitration to send message on bus**
  - Collisions, token passes, poling messages
- **Ability of nodes to accept/send data**



[Dean]

Review

- **Another look at workloads *vs.* delivery times**
  - Periodic *vs.* aperiodic
  - Worst case is what matters for most real time systems
- **Network capacity**
  - Plumbing level analysis – do all the bits fit?
  - Efficiency
- **Exact CAN schedulability calculations**
  - The longer you wait, the longer you have to wait with prioritized, periodic messages
  - This looks *a whole lot* like interrupt latency from 18-348
    - There was a reason those equations were important – they show up any time you have non-preemptable tasks!