

# 12

# CAN Performance

**18-649 Distributed Embedded Systems**

**Philip Koopman**

**October 10, 2011**

**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 last lecture)
- Protocol performance, especially CAN

## ◆ Where we're going next:

- Embedded Product Economics
- Scheduling
- Spring Break(!)
- Project 7 is worth 10% of course grade
  - 10 point bonus can give 1% of course grade; strict requirements to earn it
- First half of the course – How to make good distributed+embedded systems
- Second half of the course – How to make them better (dependable; safe; usable; ...)

# Summary of Requirements Churn (Ch 9)

---

## ◆ 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

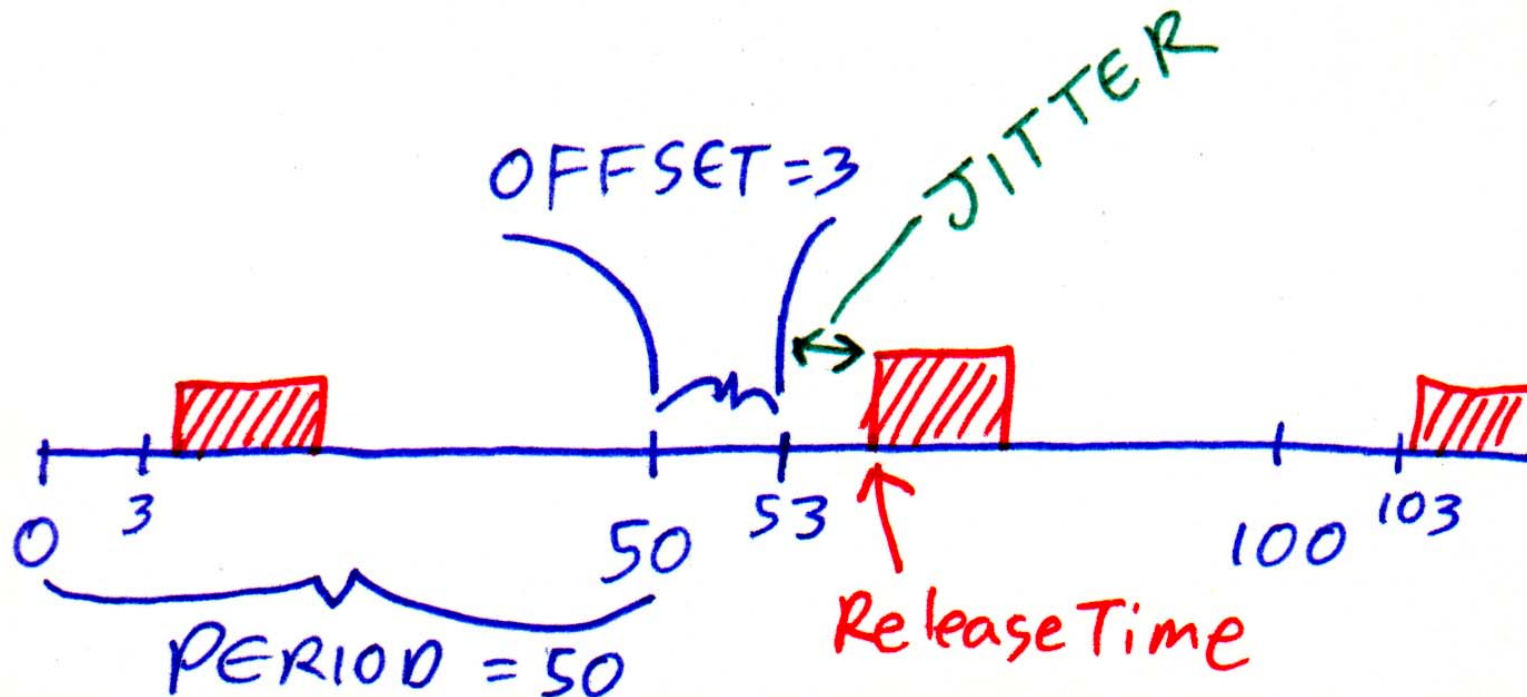
- ◆ “SAE Standard Workload” (subset of 53 messages) V/C = Vehicle Controller

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<sub>n</sub> = (n\*period) + offset + jitter ; assuming perfect time precision



# Sporadic Messages (Aperiodic; ~Exponential)

## ◆ Asynchronous messages

- External events
- Often Poisson processes (exponential inter-arrival times)

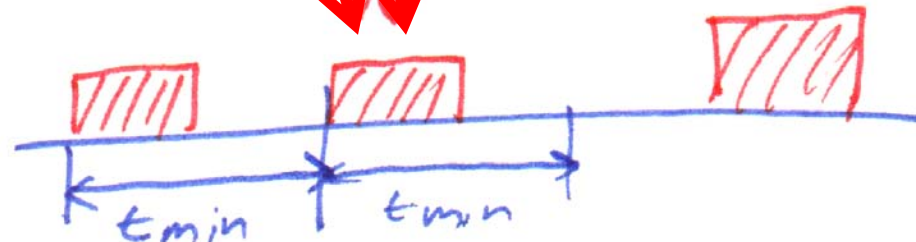
## ◆ Sporadic message timing properties

- 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

**APERIODIC**



**SPORADIC**



# 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 happens if a burst hits all at once?
  
  - Sustained maximum capacity
    - How many bits can you send with max constant transmitter demand?

# 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

---

## ◆ 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; others aren’t subject to stuffing

$$\#bits = \left( \left\lfloor \frac{54 + 8s_m}{4} \right\rfloor + 67 + 8s_m \right) \quad [\text{Ellims02}]$$

# Message Use Efficiency

---

## ◆ How many messages are actually used?

- In any token/pollled 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

# Network Capacity

---

## ◆ Instantaneous Peak capacity/demand

- This is a degenerate case – when a message is being sent you're at instantaneous peak capacity
- BUT, there is a fundamental truth here – only one message can go at a time!

## ◆ Sustained maximum capacity (data payload bytes or messages/sec)

- How many messages can be sent per unit time?
  - Assuming some mix of payload length
  - Assuming no network noise/data loss
  - Making assumptions of favorable transmitter patterns
- Increasing efficiency increases sustained maximum capacity
  - Avoid collisions
  - Avoid non-data-bearing messages
  - Minimize per-message overhead

# 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**

# Sustained Peak Demand

---

- ◆ For critical systems, you have to plan on the worst case!
- ◆ Sustained Peak demand is based on mean periods + max arrival rates
  - Periodic messages – at stated period
  - Aperiodic messages – at minimum arrival intervals
    - 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

# 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:**
  - 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

# 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)
- ◆ **Note on Deadline Monotonic Scheduling**
  - This is a generalized way to meet real time deadlines with prioritized tasks
  - Sort messages by priority order
  - Shortest period gets highest priority
  - More about this in the real time scheduling lecture

# Abbreviated “Standard” Automotive Workload

<i>Signal Number</i>	<i>Signal Description</i>	<i>Size /bits</i>	<i>J /ms</i>	<i>Period /ms</i>	<i>Periodic /Sporadic</i>	<i>Deadline /ms</i>	<i>From</i>	<i>To</i>
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	Contactora	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

# Sort For Deadline Monotonic Scheduling

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

# Workload Bandwidth Consumption

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

<i>Signal Number</i>	<i>Signal Description</i>	<i>Size /bits</i>	<i>J /ms</i>	<i>Period /ms</i>	<i>Periodic /Sporadic</i>	<i>Deadline /ms</i>	<i>From</i>	<i>bits/ message</i>	<i>bits/ second</i>
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	Contactora	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 =  $20 \times 90 = 1800$  bits if all messages are released simultaneously
  - $1800 \text{ bits} / 5 \text{ msec} = 360,000 \text{ bits/sec}$
  - 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)

<i>Signal Description</i>	<i>Size /bits</i>	<i>Deadline /ms</i>	<i>From</i>	<i>bits/ message</i>	<i>bits/ second</i>
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
Contactora	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

- Result is schedulable at ~125 Kbps (note that deadlines are harmonic)

<i>Signal Number</i>	<i>Priority</i>	<i>Signal Description</i>	<i>Size /bits</i>	<i>J /ms</i>	<i>Period /ms</i>	<i>Periodic /Sporadic</i>	<i>Deadline /ms</i>	<i>From</i>	<i>bits/ message</i>	<i>bits/ second</i>
7	<b>1</b>	Accel_Posn	8	0.1	5	P	5	Driver	90	18000
8	<b>2</b>	Brake_Master	8	0.1	5	P	5	Brakes	90	18000
9	<b>3</b>	Brake_Line	8	0.2	5	P	5	Brakes	90	18000
11	<b>4</b>	Trans_Clutch	8	0.1	5	P	5	Trans	90	18000
14	<b>5</b>	Contactora	4	0.1	50	S	5	Battery	90	18000
15	<b>6</b>	Key_Run	1	0.2	50	S	20	Driver	90	4500
16	<b>7</b>	Key_Start	1	0.3	50	S	20	Driver	90	4500
17	<b>8</b>	Accel_Switch	2	0.4	50	S	20	Driver	90	4500
18	<b>9</b>	Brake_Switch	1	0.3	20	S	20	Brakes	90	4500
19	<b>10</b>	Emer_Brake	1	0.5	50	S	20	Driver	90	4500
20	<b>11</b>	Shift_Lever	3	0.6	50	S	20	Driver	90	4500
1	<b>12</b>	T_Batt_V	8	0.6	100	P	100	Battery	90	900
2	<b>13</b>	T_Batt_C	8	0.7	100	P	100	Battery	90	900
4	<b>14</b>	A_Batt_V	8	0.8	100	P	100	Battery	90	900
6	<b>15</b>	A_Batt_C	8	0.9	100	P	100	Battery	90	900
10	<b>16</b>	Trans_Lube	8	0.2	100	P	100	Trans	90	900
12	<b>17</b>	Speed	8	0.4	100	P	100	Brakes	90	900
3	<b>18</b>	T_Batt_Tave	8	1.0	1000	P	1000	Battery	90	90
5	<b>19</b>	T_Batt_Tmax	8	1.1	1000	P	1000	Battery	90	90
13	<b>20</b>	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	Contactora	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	} Driver_msg
16	Key_Start	1	0.3	50	S	20	Driver	90	4500	
17	Accel_Switch	2	0.4	50	S	20	Driver	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	} Batt_msg1
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	
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	} Batt_msg2
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	
								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	Contactora	4	0.1	50	S	5	Battery	90	18000
2	<b>8+9</b>	<b>Brake_msg</b>	<b>16</b>	<b>0.1</b>	<b>5</b>	<b>P</b>	<b>5</b>	<b>Brakes</b>	<b>100</b>	<b>20000</b>
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	<b>15-17,19-20</b>	<b>Driver_msg</b>	<b>8</b>	<b>0.2</b>	<b>50</b>	<b>S</b>	<b>20</b>	<b>Driver</b>	<b>90</b>	<b>4500</b>
7	<b>1,2,4,6</b>	<b>Batt_msg1</b>	<b>32</b>	<b>0.6</b>	<b>100</b>	<b>P</b>	<b>100</b>	<b>Battery</b>	<b>120</b>	<b>1200</b>
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	<b>3,5,13</b>	<b>Batt_msg2</b>	<b>17</b>	<b>1.0</b>	<b>1000</b>	<b>P</b>	<b>1000</b>	<b>Battery</b>	<b>110</b>	<b>110</b>
								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?

# 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*

## ◆ 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)

Priority	Signal Description	J /ms	Deadline /ms	From	bits/message	Bits sent within n msec			
						5	20	100	1000
1	Contactora	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)

# Understanding Message Delay

## ◆ Can we get a better understanding of worst case delay?

$$W_m = B_m + \sum_{\forall k \in hp(m)} \left\lceil \frac{W_m + J_k + \tau_{bit}}{T_k} \right\rceil C_k$$

[Ellims02]

- Figures out how many periods for each message at higher priority (hp)
  - B = max blocking delay; w = queueing time; J = jitter; T = periodicity; C = transmission time
  - Yes, this is a recursive equation! (The longer you wait, the longer you have to wait)
  - Latency =  $W_m$  + message length
- For Contactor (highest priority message)
    - (Math from Ellims & Tindell papers); assume 125Kbps = 125,000 bits/sec
    - B = 120 bit blocking message possible (Batt\_msg1)
    - $W_m = 0.1$  msec jitter from result available until queued for transmission
      - » (round up to nearest 8 usec bit time = 0.104 msec = 13 bit times)
    - 90 bits delivery
    - No pre-emption possible, so summation term is zero
  - Latency = 210 bits + 0.104 msec = 1.784 msec

# Why Bother With All This Stuff?

---

- ◆ **Deadline monotonic scheduling (later lecture) is a general-purpose tool**
  - You can beat deadline monotonic in some situations
- ◆ **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
  - Doesn't include retransmissions/lost message effects
- ◆ **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

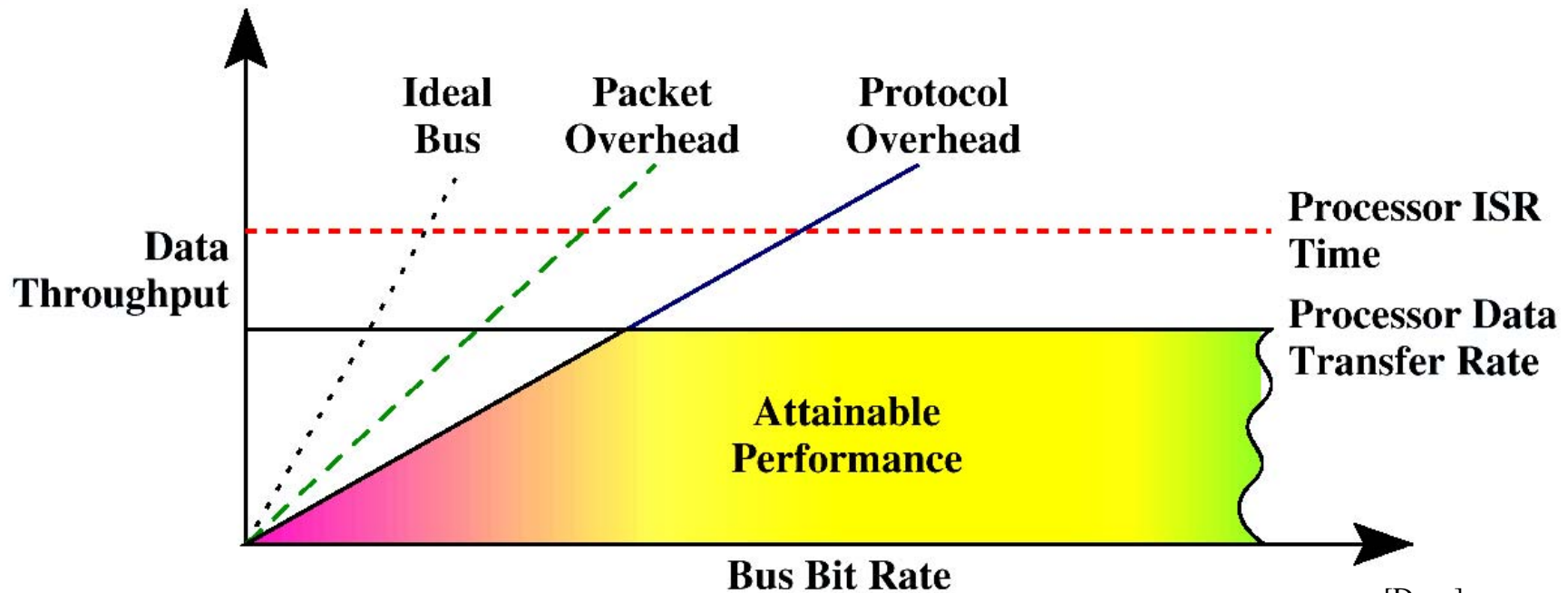
- No point scheduling tasks that produce low priority messages early in a cycle ... they'll just have to wait to transmit anyway
- Might even intentionally add some delay to spread tasks out in cycle

## ◆ 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, polling messages
- ◆ **Ability of nodes to accept/send data**



# Review

---

## ◆ Another look at workloads *vs.* delivery times

- Periodic *vs.* aperiodic
- Best case *vs.* worst case *vs.* expected case

## ◆ 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!

## ◆ **ADVANCED ELEVATOR lecture in time available this week**