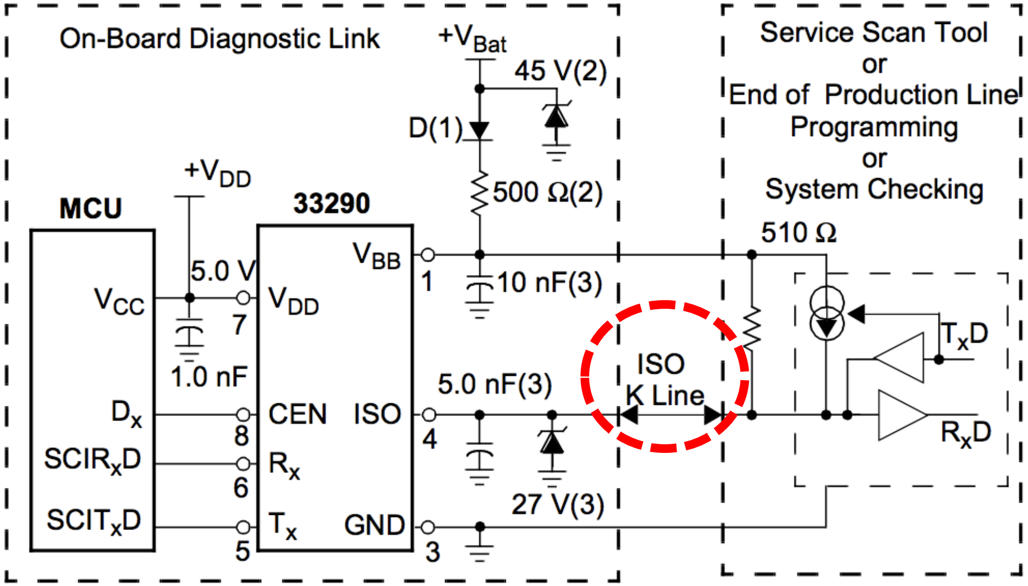
Vehicular Networks [C2X]

Part 1: In-Car Networking

Protocols: K-Line, CAN, and LIN

The K-Line Bus

- The K-Line Bus
  - Industry standard of the 80s, much later standardized as ISO 9141
  - Numerous variants exist (esp. upwards of Link Layer)
  - Lecture focuses on ISO 14230: The KWP 2000 (Keyword Protocol)
  - Specifies Physical and Link layers
  - Bidirectional bus, communicating over 1 wire (the **K Line**)

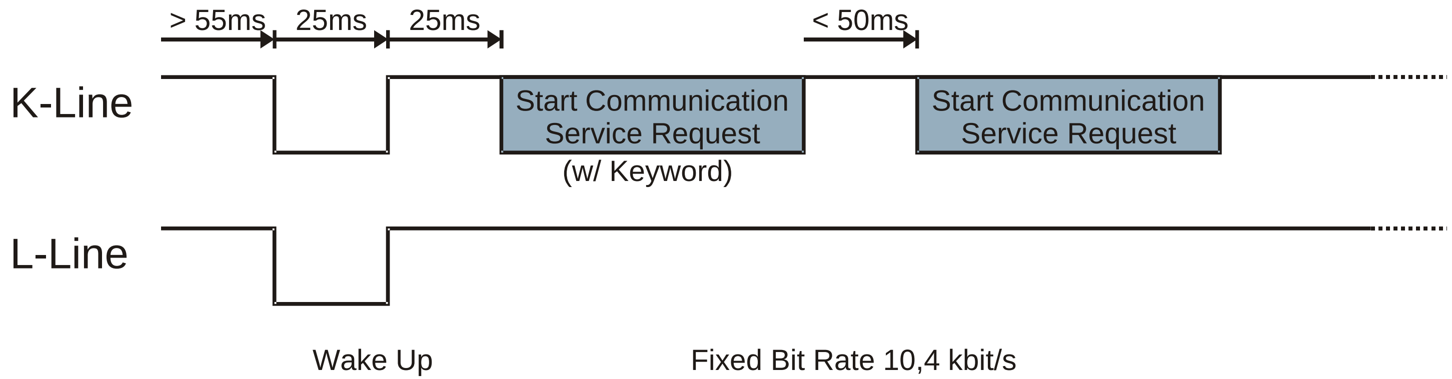


The K-Line Bus

- The K-Line Bus (contd.)
  - Optional: additional unidirectional **L Line**
    - Allows mixed networks (using only K Line / using both K+L Line)
  - Mostly used for connecting ECU ⇔ Tester, seldom ECU ⇔ ECU
  - Logic levels are relative to on board voltage (< 20% and > 80%)
  - Bit transmission compatible to UART (Universal Asynchronous Receiver Transmitter): 1 start bit, 8 data bits, 1 stop bit, optional parity bit
  - Bit rate 1.2 kBit/s ... 10.4 kBit/s
    - Dependent on ECU, not Bus
    - Master must be able to handle multiple bit rates

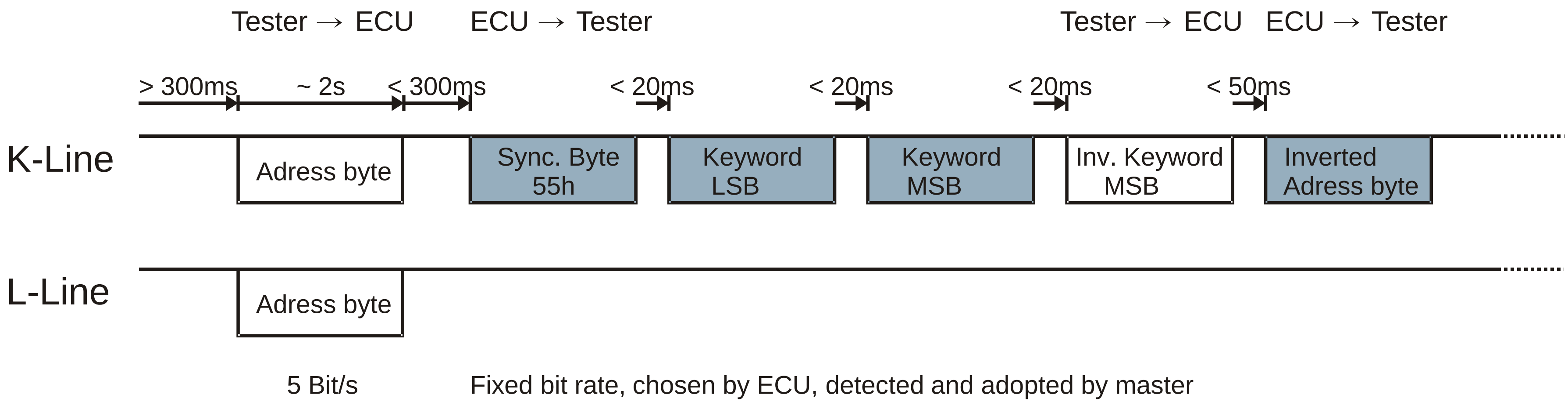
The K-Line Bus

- Protocol
  - Connection establishment (2 variants)
    - Fast init (100 ms, Bitrate always 10,4 kBit/s)
      - Master sends *Wake Up* pattern (25 ms low, 25 ms pause)
      - Master sends *Start Communication Request,* includes dest address
      - ECU answers with keyword, after max. 50 ms
      - Keyword encodes supported protocol variants
        takes values from 2000 .. 2031 (KWP 2000)



The K-Line Bus

- Protocol
  - Connection establishment (2 variants)
    - 5 Baud init
      - Master sends destination address (using 5 Bit/s)
      - ECU answers: 0x55 (01010101), keyword low Byte, keyword high Byte (with desired data rate)
      - Master derives bit rate from pattern, sends Echo (inv. High Byte)
      - ECU sends Echo (inv. Destination address)



The K-Line Bus

- Protocol
  - Communication always initiated by master
    - Master sends Request, ECU sends Response
  - Addressing
    - Address length is 1 Byte
    - Either physical address (identifies specific ECU)
    - or logical address (identifies class of ECU) e.g., engine, transmission, ...
    - Differentiated via format byte
  - Duration of single transmission at 10.4 kBit/s
    - best case: 250 ms, worst case 5.5s
    - i.e., application layer data rate < 1 KB/s

The K-Line Bus

- Protocol header
  - Format Byte
    - Encodes presence and meaning of address bytes
    - Short packet length can be encoded in format byte; length byte then omitted
  - Destination address
  - Source address
  - Length
  - Payload
    - Up to 255 Byte
    - First Byte: Service Identifier (SID)
  - Checksum
    - Sum of all Bytes (mod 256)

0 .. 7	8 .. 15
Format byte	Destination
Source	Length
Payload...	
...	Checksum

The K-Line Bus

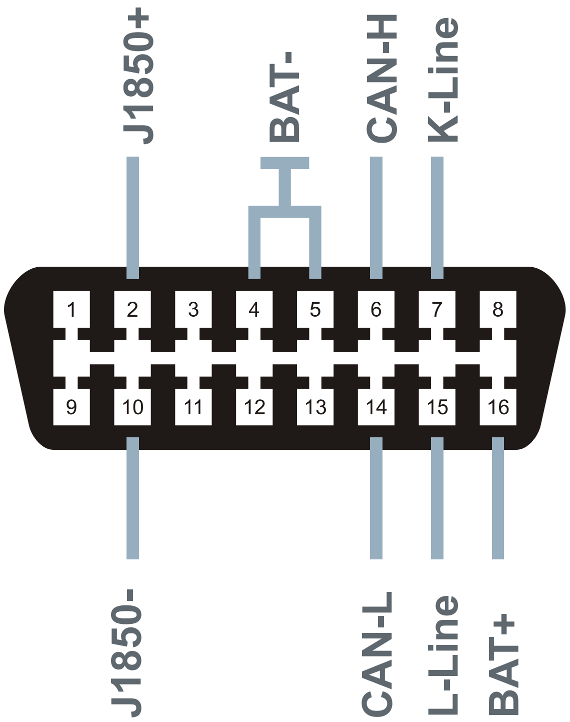
- Service Identifiers
  - Standard Service Identifiers
    - Session Initialization and teardown
      - 0x81h Start Communication Service Request
      - 0x82h Stop Communication Service Request
    - Configuring protocol timeouts
      - 0x83h Access Timing Parameter Request (optional)
  - Other SIDs are vendor defined
    - Passed on (unmodified) to application layer
    - Typical use: two SIDs per message type
      - First SID: Positive reply
      - Second: Negative reply

The K-Line Bus

- Error handling
  - If erroneous signal arrives
    - ECU ignores message
    - Master detects missing acknowledgement
    - Master repeats message
  - If invalid data is being sent
    - Application layer sends negative reply
    - Master / ECU can react accordingly

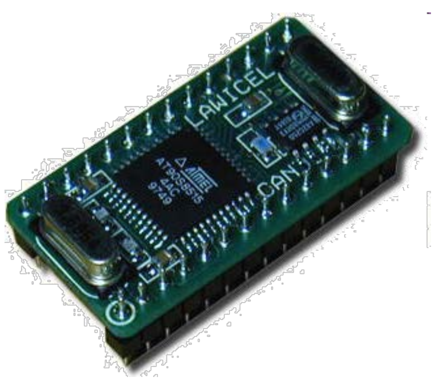
The K-Line Bus

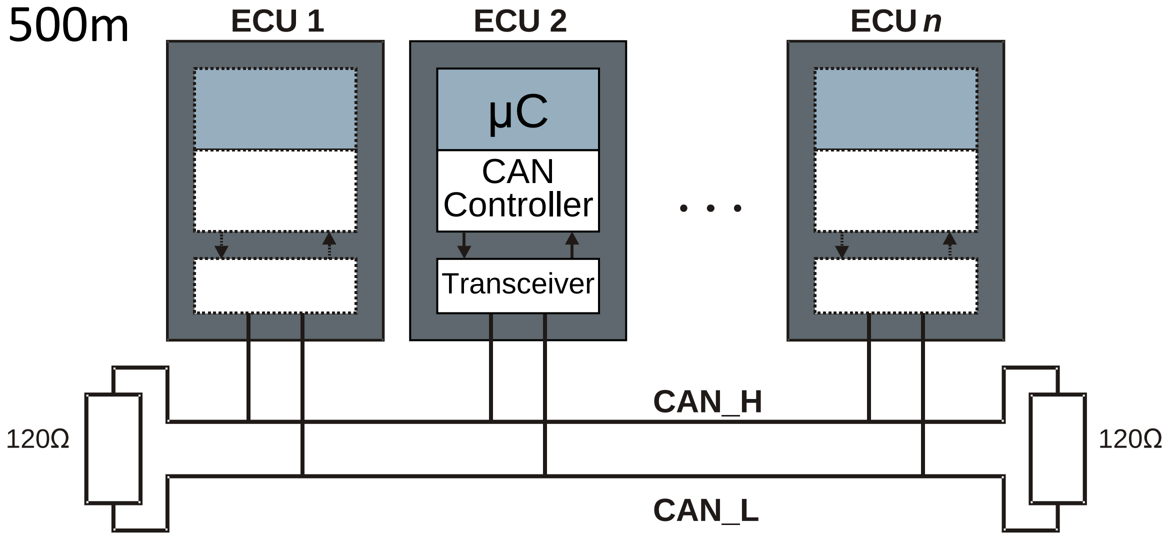
- Use in On Board Diagnostics (OBD)
  - OBD uses stricter protocol variant
  - Bit rate fixed to 10.4 kBit/s
  - No changes in timing
  - Header no longer variable
    - Length byte never included
    - Address always included
  - Max. Message length is 7 Byte
  - Shall use
    logical addressing by tester,
    physical addressing by ECUs



The CAN Bus

- The CAN Bus
  - „Controller Area Network“ (1986)
  - Network topology: Bus
  - Two signal levels
    - low (dominant), high (recessive)
  - Up to 110 nodes
    - Limited by PHY layer
  - At 125 kBit/s: max. 500m



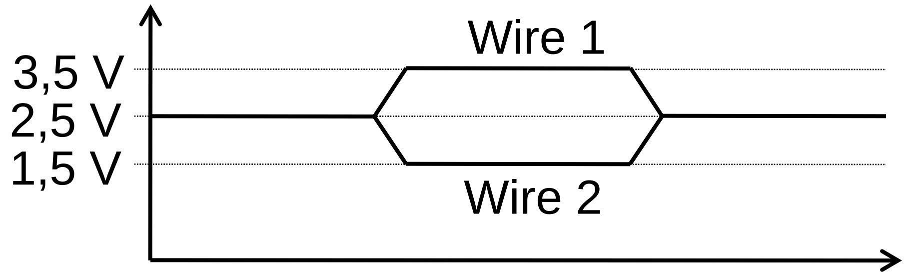


The CAN Bus

- The CAN Bus
  - ISO 11898
    - Low Speed CAN (up to 125 kBit/s)
    - High Speed CAN (up to 1 MBit/s)
  - Specifies OSI layers 1 and 2
    - Higher layers not standardized by CAN, covered by additional standards and conventions
    - E.g., CANopen
  - Random access, collision free
    - CSMA/CA with Bus arbitration
  - Message oriented
  - Does not use destination addresses
    - Implicit Broadcast/Multicast

The CAN Bus

- Physical layer (typical)
  - High Speed CAN
    - 500 kBit/s
    - Twisted pair wiring



    - Branch lines max. 30 cm
    - Terminating resistor mandated (120 Ω)
    - Signal swing 2 V
    - Error detection must happen within one Bit's time
      ⇨ bus length is limited:

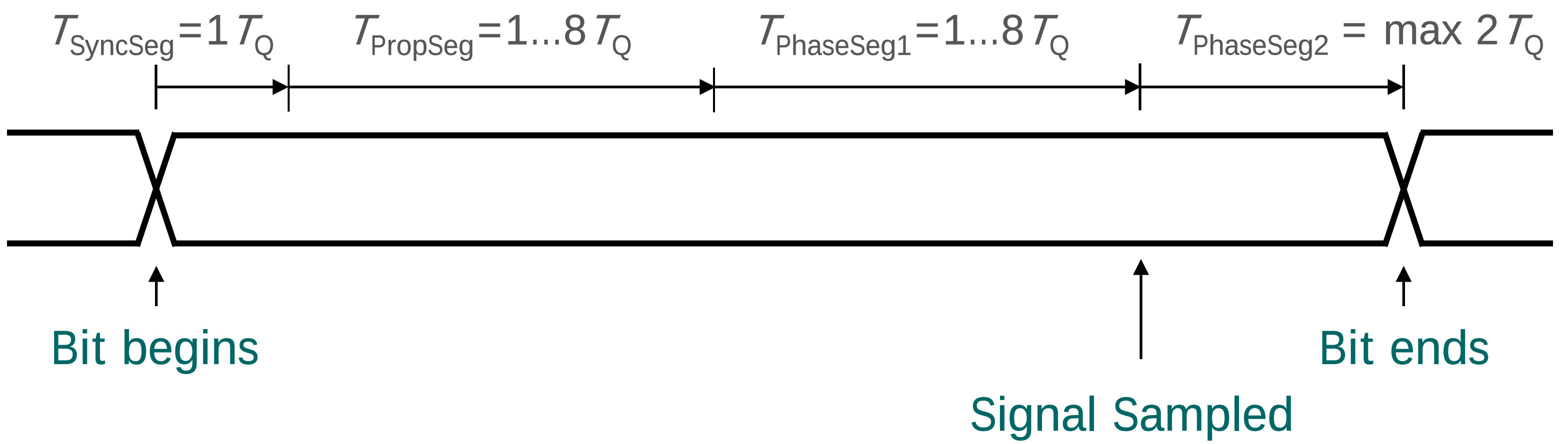
$$l \leq 50\,m \cdot \frac{1\,MBit/s}{data\,rate}$$

The CAN Bus

- Physical layer (typical)
  - Low Speed CAN
    - Up to 125 kBit/s
    - Standard two wire line suffices
    - No restriction on branch lines
    - Terminating resistors optional
    - Signal swing 5 V
  - Single Wire CAN
    - 83 kBit/s
    - One line vs. ground
    - Signal swing 5 V

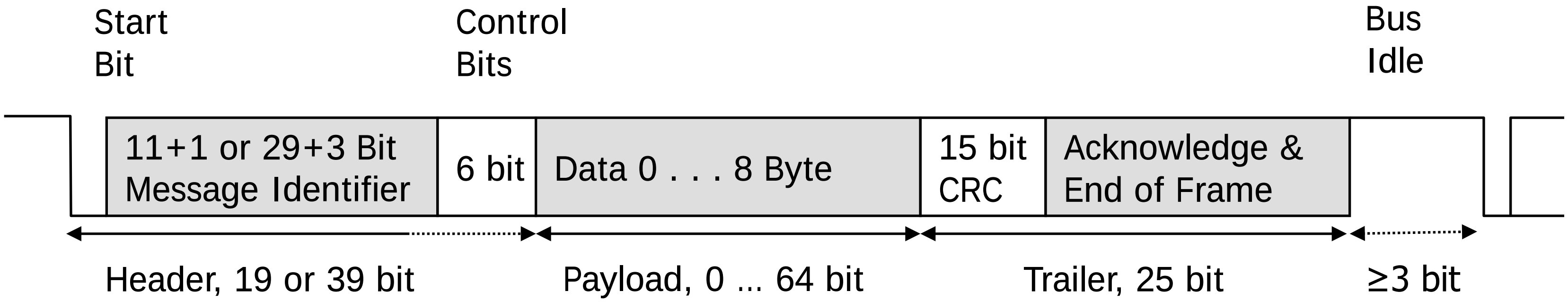
CAN in Vehicular Networks

- Bit Timing
  - Times derived from clock time (Quantum) T_Q
  - Bit time T_{bit} consists of sync segment $T_{SyncSeg}$, propagation segment $T_{PropSeg}$, phase segments $T_{PhaseSeg1}$, $T_{PhaseSeg2}$ (can be adapted by controller for synchronization)
  - $T_{SyncSeg}$+ $T_{PropSeg}$ must be longer than 2x propagation delay
  - Signal sampled between $T_{PhaseSeg1}$ and $T_{PhaseSeg2}$
  - Standard recommends, e.g. at 500 kbps, T_Q = 125 ns, T_{bit} =16 T_Q



CAN in Vehicular Networks

- Address-less communication
  - Messages carry 11 Bit or 29 Bit message identifier
  - Stations do not have an address, Frames do not contain one
  - Stations use message identifier to decide whether a message is meant for them
  - Medium access using CSMA/CA with bitwise arbitration
  - Link layer uses 4 frame formats
    Data, Remote (request), Error, Overload (flow control)
  - Data frame format:



CAN in Vehicular Networks

- CSMA/CA with bitwise arbitration (CSMA/CR)
  - Avoids collisions by priority-controlled bus access
  - Each message contains identifier corresponding to its priority
  - Identifier encodes "0" dominant and "1" recessive:
    concurrent transmission of "0" and "1" results in a "0"
  - Bit stuffing: after 5 identical Bits one inverted Stuff-Bit is inserted (ignored by receiver)
  - When no station is sending the bus reads "1" (recessive state)
  - Synchronization happens on bit level,
    by detecting start bit of sending station

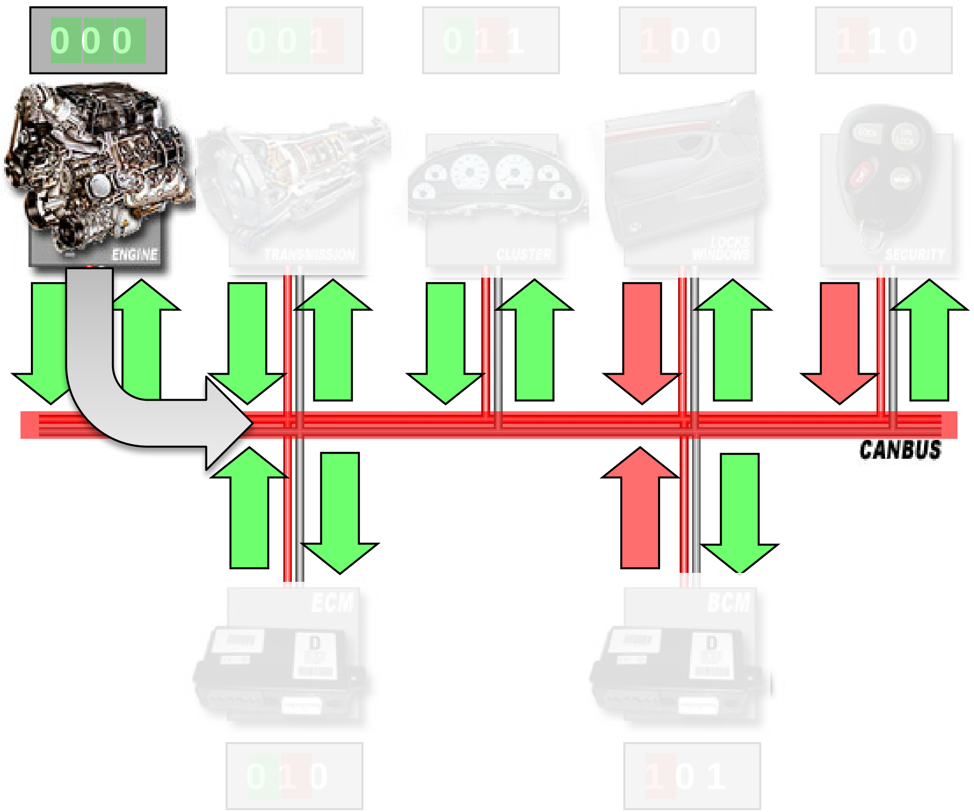
CAN in Vehicular Networks

- CSMA/CA with bitwise arbitration (CSMA/CR)
  - Wait for end of current transmission
    - wait for 6 consecutive recessive Bits
  - Send identifier (while listening to bus)
  - Watch for mismatch between transmitted/detected signal level
    - Means that a collision with a higher priority message has occurred
    - Back off from bus access, retry later

  - Realization of non-preemptive priority scheme
  - Real time guarantees for message with highest priority
    - i.e., message with longest “0”-prefix

CAN in Vehicular Networks

- CSMA/CA with bitwise arbitration (CSMA/CR)
  - Example (recall: “0” dominant, “1” recessive)

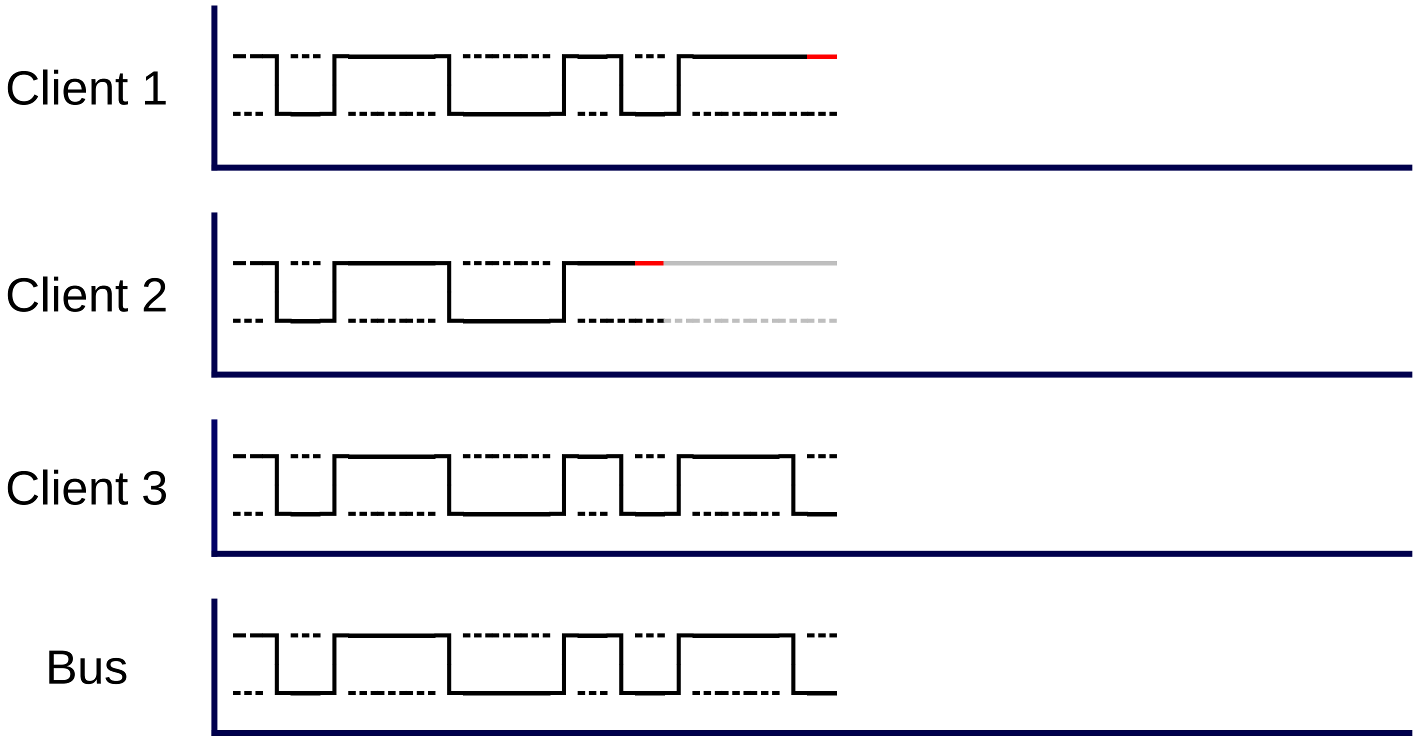


The CAN Bus

- CSMA/CA with bitwise arbitration (CSMA/CR)
  - Client 2 recognizes bus level mismatch, backs off from access

The CAN Bus

- CSMA/CA with bitwise arbitration (CSMA/CR)
  - Client 1 recognizes bus level mismatch, backs off from access

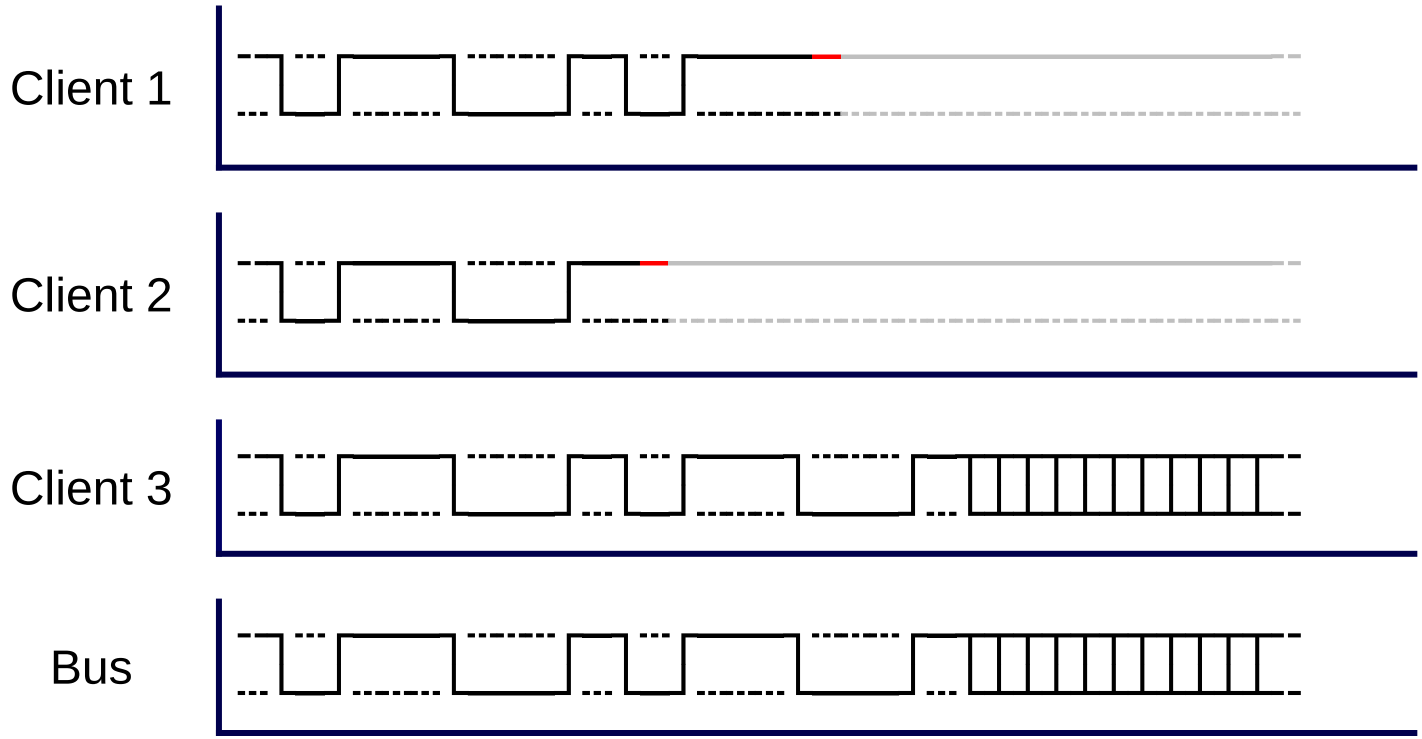


The CAN Bus

- CSMA/CA with bitwise arbitration (CSMA/CR)
  - Client 3 wins arbitration

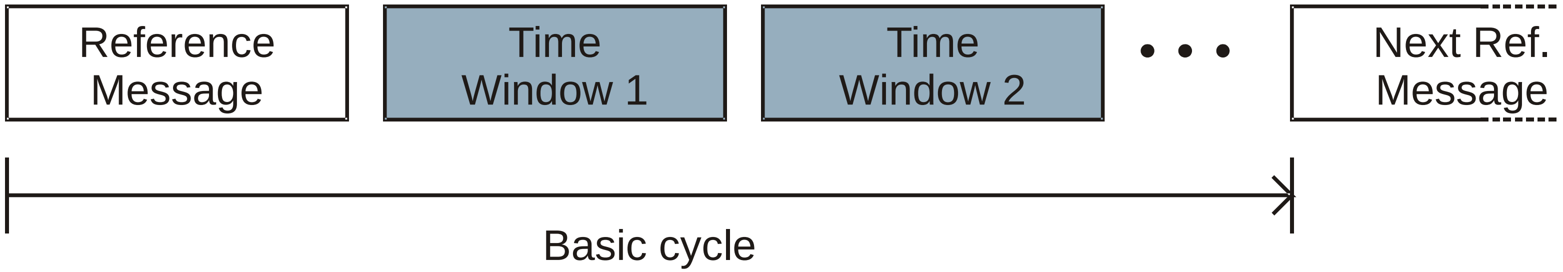
The CAN Bus

- CSMA/CA with bitwise arbitration (CSMA/CR)
  - Client 3 starts transmitting data



The CAN Bus: TTCAN

- Time-Triggered CAN (TTCAN)
  - ISO 11898-4 extends CAN by TDMA functionality
  - Solves non-determinism of regular CAN
    - Improves on mere “smart” way of choosing message priorities
  - One node is dedicated “time master” node
  - Periodically sends reference messages starting “basic cycles”
  - Even if time master fails, TTCAN keeps working
    - Up to 7 fallback nodes
    - Nodes compete for transmission of reference messages
    - Chosen by arbitration



The CAN Bus: TTCAN

- TTCAN Basic Cycle
  - Basic cycle consists of time slots
    - Exclusive time slot
      - Reserved for dedicated client
    - Arbitration time slot
      - Regular CAN CSMA/CA with bus arbitration
  - Structure of a basic cycle arbitrary, but static
  - CAN protocol used unmodified
    - ➔ Throughput unchanged
  - TTCAN cannot be seen replacing CAN for real time applications
    - Instead, new protocols are being used altogether (e.g., FlexRay)

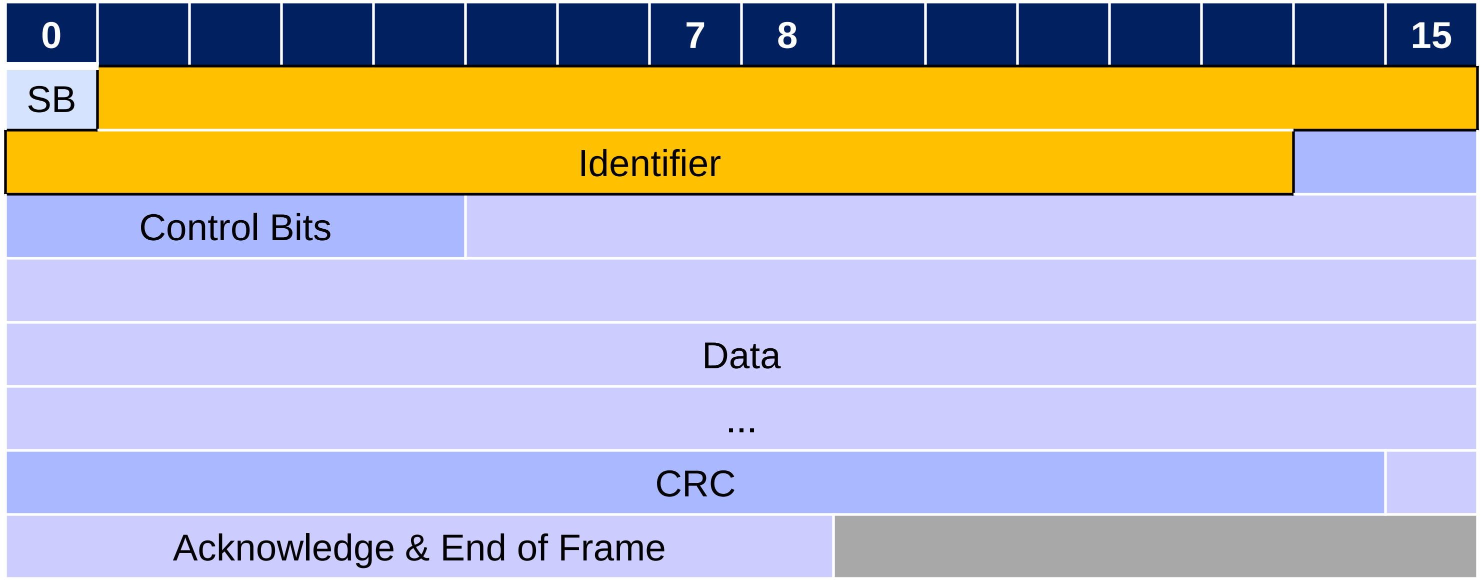
The CAN Bus

- Message filtering
  - Acceptance of messages determined by message identifier
  - Uses two registers
    - Acceptance Code (bit pattern to filter on)
    - Acceptance Mask (“0” marks relevant bits in acceptance code)

Bit	10	9	8	7	6	5	4	3	2	1	0
Acceptance Code Reg.	0	1	1	0	1	1	1	0	0	0	0
Acceptance Mask Reg.	1	1	1	1	1	1	1	0	0	0	0
Resulting Filter Pattern	**0**	**1**	**1**	**0**	**1**	**1**	**1**	**X**	**X**	**X**	**X**

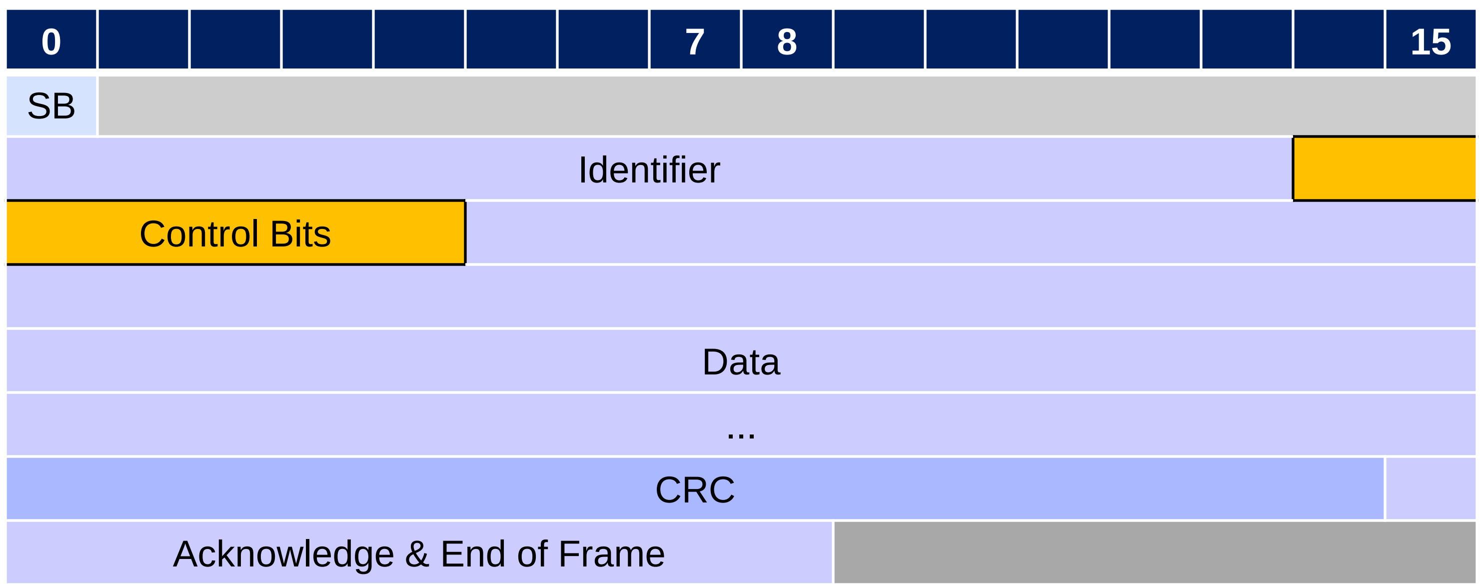
The CAN Bus

- Data format
  - NRZ
  - Time synchronization using start bit and stuff bits (stuff width 5)
  - Frame begins with start bit
  - Message identifier 11 Bit (CAN 2.0A), now 29 Bit (CAN 2.0B)



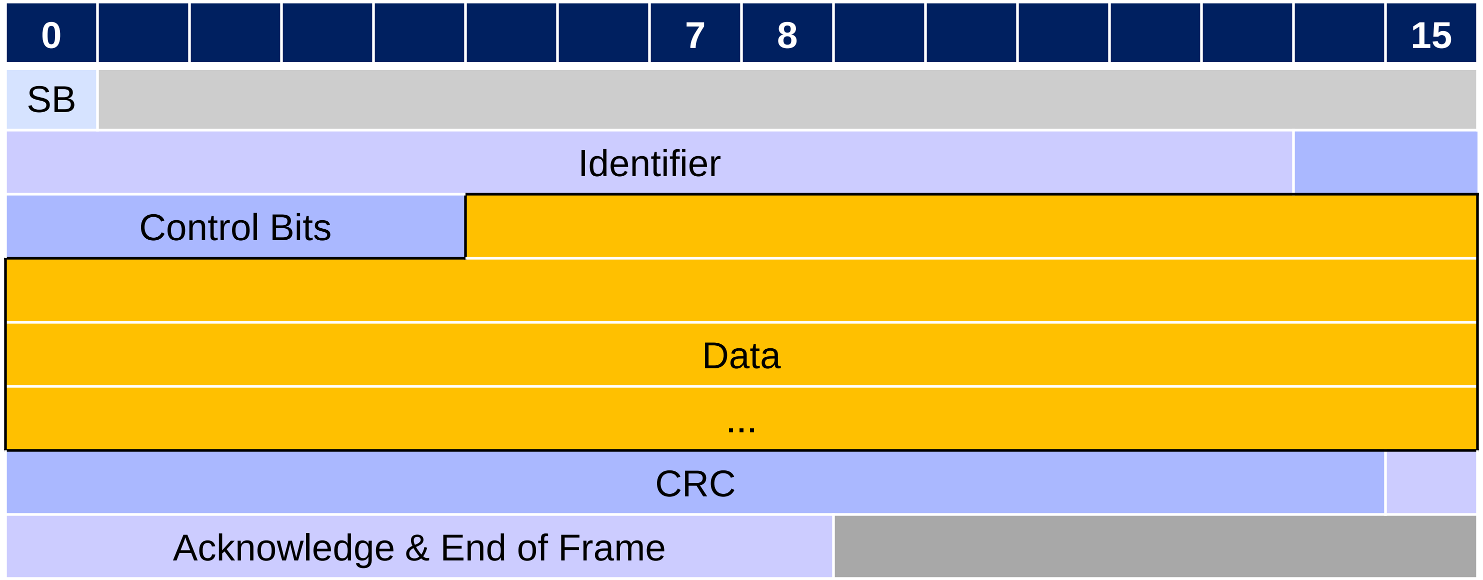
The CAN Bus

- Data format
  - Control Bits
    - Message type (Request, Data, Error, Overload)
    - Message length
    - ...



The CAN Bus

- Data format
  - Payload
    - Restriction to max. 8 Byte per message
    - Transmission time at 500 kBit/s: 260 µs (using 29 Bit ID)
    - i.e., usable data rate 30 kBit/s



The CAN Bus

- Error detection (low level)
  - Sender checks for unexpected signal levels on bus
  - All nodes monitor messages on the bus
    - All nodes check protocol conformance of messages
    - All nodes check bit stuffing
  - Receiver checks CRC
  - If any(!) node detects error it transmits error signal
    - 6 dominant Bits with no stuffing
  - All nodes detect error signal, discard message

The CAN Bus

- Error detection (high level)
  - Sender checks for acknowledgement
    - Receiver transmits dominant “0”
      during ACK field of received message
  - Automatic repeat of failed transmissions
  - If controller finds itself causing too many errors
    - Temporarily stop any bus access
  - Remaining failure probability ca. 10^{-11}

The CAN Bus: Transport Layers

- Not covered by ISO 11898 (CAN) standards
  - Fragmentation
  - Flow control
  - Routing to other networks

- Add transport layer protocol
  - ISO-TP
    - ISO 15765-2
  - TP 2.0
    - Industry standard
  - …

The CAN Bus: ISO-TP

- ISO-TP: Header
  - Optional: 1 additional address Byte
    - Regular addressing
      - Transport protocol address completely in CAN message ID
    - Extended addressing
      - Uniqueness of addresses despite non-unique CAN message ID
      - Part of transport protocol address in CAN message ID, additional address information in first Byte of TP-Header
  - 1 to 3 PCI Bytes (Protocol Control Information)
    - First high nibble identifies one of 4 types of message
    - First low nibble and addl. Bytes are message specific

0	1		2	3	4	5	6	7
(opt) Addl. Address	PCI high	PCI low	(opt) Addl. PCI Bytes		Payload			

The CAN Bus: ISO-TP

- ISO-TP: Message type “Single Frame”
  - 1 Byte PCI, high nibble is 0
  - low nibble gives number of Bytes in payload
  - PCI reduces frame size from 8 Bytes to 7 (or 6) Bytes, throughput falls to 87.5% (or 75%, respectively)
  - No flow control

<table>
<tr><th colspan="2">0</th><th>1</th><th>2</th><th>3</th><th>4</th><th>5</th><th>6</th><th>7</th></tr>
<tr><td>0</td><td>Len</td><td colspan="7">Payload</td></tr>
</table>

<table>
<tr><th>0</th><th colspan="2">1</th><th>2</th><th>3</th><th>4</th><th>5</th><th>6</th><th>7</th></tr>
<tr><td>(Address)</td><td>0</td><td>Len</td><td colspan="6">Payload</td></tr>
</table>

The CAN Bus: ISO-TP

- ISO-TP: Message type „First Frame“
  - 2 Bytes PCI, high nibble is 1
  - low nibble + 1 Byte give number of Bytes in payload
  - After First Frame, sender waits for Flow Control Frame

0	1		2	3	4	5	6	7
(Address)	1	Len		Payload				

- ISO-TP: Message type „Consecutive Frame“
  - 1 Byte PCI, high nibble is 2
  - low nibble is sequence number SN (counts upwards from 1)
    - Application layer can detect packet loss
  - No additional error detection at transport layer

0	1		2	3	4	5	6	7
(Address)	2	SN	Payload					

The CAN Bus: ISO-TP

- ISO-TP: Message type „Flow Control Frame“
  - 3 Bytes PCI, high nibble is 3
  - low nibble specifies Flow State FS
  - FS=1: Clear to Send
    - Minimum time between two Consecutive Frames must be ST
    - Sender may continue sending up to BS Consecutive Frames, then wait for new Flow Control Frame
  - FS=2: Wait
    - Overload
    - Sender must wait for next Flow Control Frame
  - Byte 2 specifies Block Size BS
  - Byte 3 specifies Separation Time ST

0	1		2	3
(Address)	3	FS	BS	ST

The CAN Bus: TP 2.0

- TP 2.0
  - Connection oriented
  - Communication based on channels
  - Specifies Setup, Configuration, Transmission, Teardown

  - Addressing
    - Every ECU has unique logical address;
      additional logical addresses specify groups of ECUs
    - for broadcast und channel setup:
      logical address + offset = CAN message identifier
    - Channels use dynamic CAN message identifier

The CAN Bus: TP 2.0

- TP 2.0: Broadcast
  - Repeated 5 times (motivated by potential packet loss)
  - Fixed length: 7 Byte
  - Byte 0:
    - logical address of destination ECU
  - Byte 1: Opcode
    - 0x23: Broadcast Request
    - 0x24: Broadcast Response
  - Byte 2, 3, 4:
    - Service ID (SID) and parameters
  - Byte 5, 6:
    - Response: 0x0000
    - No response expected: alternates between 0x5555 / 0xAAAA

0	1	2	3	4	5	6
Dest	Opcode	SID, Parameter			0x55	0x55

The CAN Bus: TP 2.0

- TP 2.0: channel setup
  - Byte 0:
    - logical address destination ECU
  - Byte 1: Opcode
    - 0xC0: Channel Request
    - 0xD0: Positive Response
    - 0xD6 .. 0xD8: Negative Response
  - Byte 2, 3: RX ID
    - Validity nibble of Byte 3 is 0 (1 if RX ID not set)
  - Byte 4, 5: TX ID
    - Validity nibble of Byte 5 is 0 (1 if TX ID not set)
  - Byte 6: Application Type
    - cf. TCP-Ports

<table>
<tr><th>0</th><th>1</th><th>2</th><th colspan="2">3</th><th>4</th><th colspan="2">5</th><th>6</th></tr>
<tr><td>Dest</td><td>Opcode</td><td colspan="2">RX ID</td><td>V</td><td colspan="2">TX ID</td><td>V</td><td>App</td></tr>
</table>

The CAN Bus: TP 2.0

- TP 2.0: channel setup (II)
  - Opcode 0xC0: Channel Request
    - TX ID: CAN msg ID requested by self
    - RX ID: marked invalid
  - Opcode 0xD0: Positive Response
    - TX ID: CAN msg ID requested by self
    - RX ID: CAN msg ID of original sender
  - Opcode 0xD6 .. 0xD8: Negative Response
    - Reports errors assigning channel (temporary or permanent)
    - Sender may repeat Channel Request
  - After successful exchange of Channel Request/Response: dynamic CAN msg IDs now assigned to sender and receiver next message sets channel parameters

0	1	2	3	4	5	6
Dest	0xC0		1	TX ID	0	App

The CAN Bus: TP 2.0

- TP 2.0: set channel parameters
  - Byte 0: Opcode
    - 0xA0: Channel Setup Request (Parameters for channel to initiator)
    - 0xA1: Channel Setup Response (Parameter for reverse channel)
  - Byte 1: Block size
    - Number of CAN messages until sender has to wait for ACK
  - Byte 2, 3, 4, 5: Timing parameters
    - E.g., minimal time between two CAN messages
- TP 2.0: misc. channel management and teardown
  - Byte 0: Opcode
    - 0xA3: Test – will be answered by Connection Setup Response
    - 0xA4: Break – Receiver discards data since last ACK
    - 0xA5: Disconnect – Receiver responds with disconnect, too

0	1	2	3	4	5
0xA0	BS	Timing			

The CAN Bus: TP 2.0

- TP 2.0: Data transmission via channels
  - Byte 0, high nibble: Opcode
    - MSB=0 – Payload
      - /AR=0 – Sender now waiting for ACK
      - EOM=1 – Last message of a block
    - MSB=1 – ACK message only (no payload)
      - RS=1 – ready for next message (➔ flow control)
  - Byte 0, low nibble
    - Sequence number
  - Bytes 1 .. 7: Payload

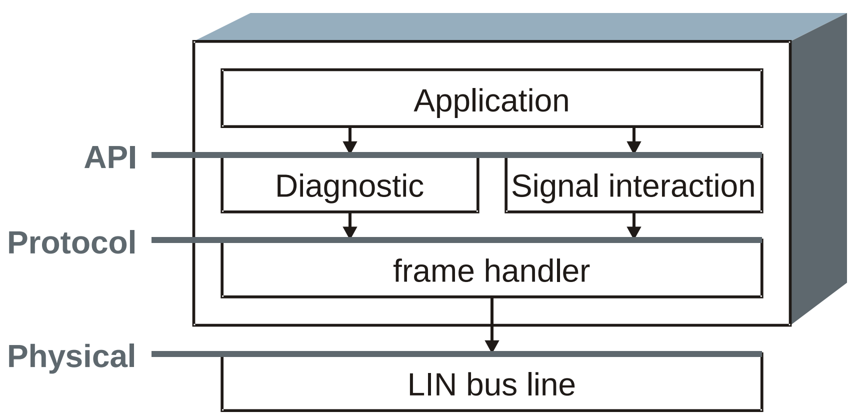
Opcode Nibble			
0	0	/AR	EOM

Opcode Nibble			
1	0	RS	1

0		1	2	3	4	5	6	7
Op	SN	Payload						

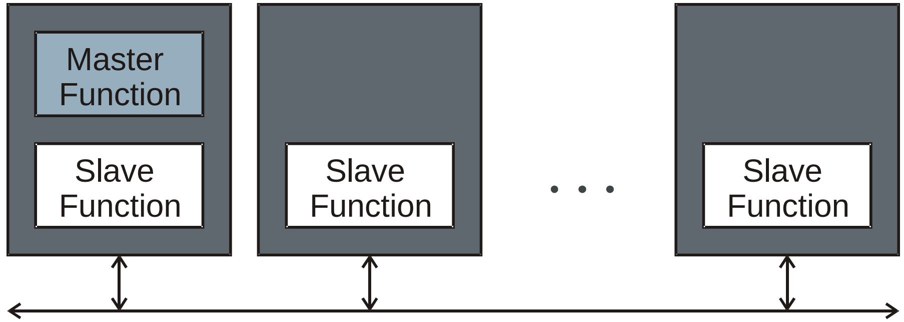
The LIN Bus

- Local Interconnect Network (LIN)
- 1999: LIN 1.0
- 2003: LIN 2.0
  - Numerous extensions
  - Backwards compatible (only)
- Goal of LIN: be much cheaper than low speed CAN
  - Only reached partway
- specifies PHY and MAC Layer, API



The LIN Bus

- Very similar to K-Line Bus
- Master-slave concept with self synchronization
  - no quartz needed
  - lax timing constraints
- LIN master commonly also part of a CAN bus
  - LIN commonly called a sub bus
- Bidirectional one-wire line, up to 20 kBit/s
- Bit transmission UART compatible
  - 1 Start Bit, 8 Data Bits, 1 Stop Bit
- Message oriented
  - No destination address

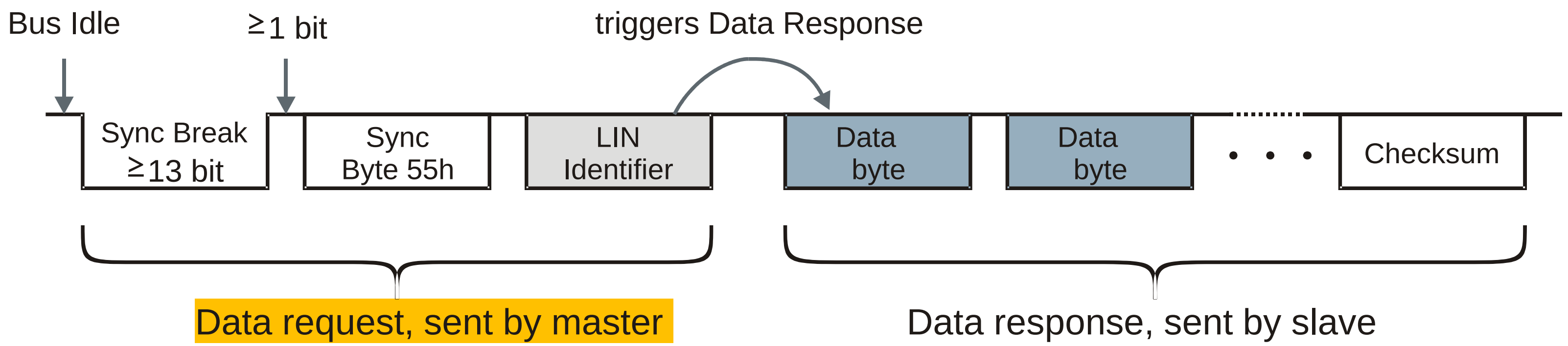


The LIN Bus

- Rudimentary error detection
  - Sender monitors bus
  - Aborts transmission on unexpected bus state
- No error correction
- Starting with LIN 2.0: Response Error Bit
  - Should be contained in periodic messages
  - Set (once) if slave detected an error in last cycle
- Static slot schedule in the master
  - "Schedule Table"
  - Determines cyclic schedule of messages transmitted by master
    → Bus timing mostly deterministic
  - Slaves do not need to know schedule
    → can be changed at run-time

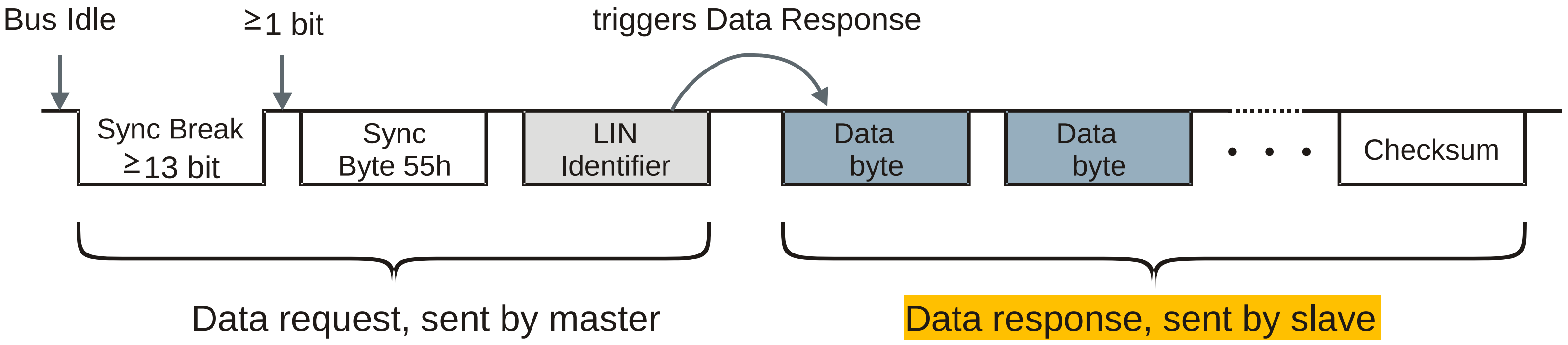
The LIN Bus

- Data request
  - Sync Break (≥13 Low Bits, 1 High Bit)
    - Not UART compliant → uniquely identifiable
  - Sync Byte 0x55 (01010101)
    - Synchronizes bit timing of slave
  - LIN Identifier (6 data Bits + 2 parity Bits)
    - Encodes response's expected message type and length
    - 0x00 .. 0x3B: application defined data types, 0x3C .. 0x3D: Diagnosis, 0x3E: application defined, 0x3F: reserved
    - Parity Bits: $I_0 \oplus I_1 \oplus I_2 \oplus I_4$ and $\neg (I_1 \oplus I_3 \oplus I_4 \oplus I_5)$



The LIN Bus

- Data response
  - Slave responds with up to 8 Bytes of data
    - LSB first, Little Endian
    - length was defined by LIN Identifier
  - Frame ends with checksum
    - LIN 1.3: Classic Checksum (only data bytes)
    - LIN 2.0: Enhanced Checksum (data bytes + Identifier)
    - Checksum is sum of all Bytes (mod 256), plus sum of all carries



The LIN Bus

- Types of requests
  - Unconditional Frame
  - Event Triggered Frame
  - Sporadic Frame
  - ...
- Unconditional Frame
  - Most simple frame type
  - Designed for periodic polling of specific data point
  - Exactly one slave answers
  - LIN is a single master system → timing of unconditional frames fully deterministic
  - Sample use case:
    - Request “did state of front left door contact change?” every 15 ms
    - Receive negative reply by front left door ECU every 15 ms

The LIN Bus

- Types of requests
  - Unconditional Frame
  - Event Triggered Frame
  - Sporadic Frame
  - ...
- Event Triggered Frame
  - Simultaneous polling of multiple slaves, slave answers if needed
  - Collisions possible (→ non-determinism), detect by corrupt. data
    - master switches to individual polling via Unconditional Frames
  - Use whenever slaves unlikely to respond
  - Sample use case:
    - Request “did state of a door contact change?” every 15 ms
    - Change in state unlikely, simultaneous change extremely unlikely

The LIN Bus

- Types of requests
  - Unconditional Frame
  - Event Triggered Frame
  - Sporadic Frame
  - ...
- Sporadic Frame
  - Sent (by master) only when needed
  - Shared schedule slot with other Sporadic Frames
  - Use whenever polling for specific data only seldom needed
  - If more than one Sporadic Frame needs to be sent, master needs to decide for one → no collision, but still non-deterministic
  - Sample use case:
    - Request „power window fully closed?“ every 15 ms
    - ...only while power window is closing

The LIN Bus

- Doing Off-Board-Diagnosis of LIN ECUs
  - Variant 1: Master at CAN bus responds on behalf of ECU on LIN
    - Keeps synchronized state via LIN messages
  - Variant 2: Master at CAN bus tunnels, e.g., KWP 2000 messages
    - Standardized protocol
    - LIN dest address is 0x3C (Byte 1 is ISO dest address)
    - Dest ECU (according to ISO address) answers with address 0x3D
    - Independent of payload, LIN frame padded to 8 Bytes
    - LIN slaves have to also support KWP 2000
    - Contradicts low cost approach of LIN
    - “Diagnostic Class” indicates level of support

Main Takeaways

- Overall
  - Design goals
  - Message orientation vs. address orientation,
  - Addressing schemes
  - Medium access
  - Flow control
  - Real time guarantees and determinism
- K-Line
  - Mainly for diagnostics
  - Transmission uses UART signaling
  - Communication using Request-Response pattern
- CAN
  - Still standard bus in vehicles
  - Message oriented
  - CSMA/CA with bitwise arbitration
    - Impact on determinism
    - TTCAN (TDMA)
  - Error detection
  - Transport layer: ISO-TP vs. TP 2.0
    - Flow control, channel concept
- LIN
  - Goals
  - Deployment as sub bus
  - Message types and scheduling
  - Determinism