RIOT and CAN

Vincent Dupont

OTA keys

RIOT Summit
September 25-26, 2017
Who am I? What is OTA keys?

Me:

- Embedded software engineer: 6 years, 3 at OTA keys
- RIOT: 1.5 year
  - Hardware support
  - Device drivers
  - Storage
  - CAN support
Who am I? What is OTA keys?

Me:
- Embedded software engineer: 6 years, 3 at OTA keys
- RIOT: 1.5 year
  - Hardware support
  - Device drivers
  - Storage
  - CAN support

OTA keys:
- Continental subsidiary: car-sharing systems
- Embedded system, backend, mobile
- Created 3 years ago, joint-venture between Continental and D’Ieteren (Belgian VW group importer)
Goal of this presentation

1. CAN bus technology
2. RIOT CAN stack
3. CAN stack usage example
1. **What is CAN?**
   - Physical Layer
   - Link layer
   - ISO-TP

2. **CAN in RIOT**

3. **Use case example: OBD**

4. **Future**
What is CAN?
Definition

CAN is a multi-master serial bus standard for connecting Electronic Control Units [ECUs] also known as nodes.
Definition

CAN is a multi-master serial bus standard for connecting Electronic Control Units [ECUs] also known as nodes.
**Physical Layer: ISO 11898-2**

**Definition**

CAN is a multi-master serial bus standard for connecting Electronic Control Units [ECUs] also known as nodes.

![Diagram of CAN bus with nodes](image)

**Recessive bus**

Dominant bit (0) bus is electrically driven, recessive (1) is not (node is open-drain)
CAN Layers

CAN Node

Microcontroller

CAN Controller

CAN Transceiver

Data Link Layer
ISO 11898-1

Medium Access Unit
(Electrical Levels)
ISO 11898-2, 3

Bus
Link Layer: ISO 11898-1

- 8 bytes per frame
- Error management
- Frames are addressed, not nodes: CAN IDentifiers
- IDs are priority (the lower, the more priority)
- 8 bytes per frame
- Error management
- Frames are addressed, not nodes: CAN IDentifiers
- IDs are priority (the lower, the more priority) → arbitration phase:

<table>
<thead>
<tr>
<th>Node</th>
<th>CAN ID</th>
<th>Data 1</th>
<th>Data 2</th>
<th>Data 3</th>
<th>Data 4</th>
<th>Priority</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>079</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>0</td>
<td>0</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Node B stops transmitting
Node C stops transmitting
- 8 bytes per frame
- Error management
- Frames are addressed, not nodes: CAN IDentifiers
- IDs are priority (the lower, the more priority) → arbitration phase:

<table>
<thead>
<tr>
<th>Node A</th>
<th>079</th>
<th>0 (SOF)</th>
<th>0</th>
<th>0</th>
<th>0</th>
<th>0</th>
<th>1</th>
<th>1</th>
<th>1</th>
<th>1</th>
<th>0</th>
<th>0</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>Node B</td>
<td>080</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Stops transmitting</td>
</tr>
<tr>
<td>Node C</td>
<td>700</td>
<td>0</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Stops transmitting</td>
</tr>
</tbody>
</table>
Link Layer: ISO 11898-1

- 8 bytes per frame
- Error management
- Frames are addressed, not nodes: CAN IDentifiers
- IDs are priority (the lower, the more priority) → arbitration phase:

| Node | CAN ID | CAN Frame |仲裁
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>079</td>
<td>0 (SOF)</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>

Node A

| Node | CAN ID | CAN Frame | arbitration
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>B</td>
<td>080</td>
<td>0</td>
<td>Stopped transmitting</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>

Node B

| Node | CAN ID | CAN Frame | arbitration
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>C</td>
<td>700</td>
<td>0</td>
<td>Stopped transmitting</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>

Node C

Vincent Dupont (OTA keys)

RIOT and CAN

RIOT Summit 2017
8 bytes per frame

Error management

Frames are addressed, not nodes: CAN IDentifiers

IDs are priority (the lower, the more priority) → arbitration phase:

<table>
<thead>
<tr>
<th>Node A</th>
<th>079</th>
<th>0 (SOF)</th>
<th>0</th>
<th>0</th>
<th>0</th>
<th>0</th>
<th>1</th>
<th>1</th>
<th>1</th>
<th>1</th>
<th>0</th>
<th>0</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>Node B</td>
<td>080</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Stops transmitting</td>
<td></td>
</tr>
<tr>
<td>Node C</td>
<td>700</td>
<td>0</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Stops transmitting</td>
<td></td>
</tr>
</tbody>
</table>
8 bytes per frame

Error management

Frames are addressed, not nodes: CAN IDentifiers

IDs are priority (the lower, the more priority) → arbitration phase:

<table>
<thead>
<tr>
<th>Node A</th>
<th>079</th>
<th>0 (SOF)</th>
<th>0</th>
<th>0</th>
<th>0</th>
<th>0</th>
<th>1</th>
<th>1</th>
<th>1</th>
<th>1</th>
<th>0</th>
<th>0</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>Node B</td>
<td>080</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>Stops transmitting</td>
</tr>
<tr>
<td>Node C</td>
<td>700</td>
<td>0</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Stops transmitting</td>
</tr>
</tbody>
</table>

Filters: receive if \((\text{can} \_\text{id} \& \text{mask}) = = \text{filter}\)
- Segmentation
- Up to 4095 bytes
- Use a pair of CAN IDs
- “Channel” between 2 nodes
Segmentation
Up to 4095 bytes
Use a pair of CAN IDs
“Channel” between 2 nodes

Header (4 bits):

<table>
<thead>
<tr>
<th>Value</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Single frame (SF)</td>
</tr>
<tr>
<td>1</td>
<td>First frame (FF)</td>
</tr>
<tr>
<td>2</td>
<td>Consecutive frame (CF)</td>
</tr>
<tr>
<td>3</td>
<td>Flow control (FC)</td>
</tr>
</tbody>
</table>
ISO-TP: ISO 15765-2, Example

Node A

First frame (length, data)

Flow control (block size, stmin)

Consecutive frame (index, data)

Consecutive frame (...)

Consecutive frame (...)

Consecutive frame (...)

Flow control (block size, stmin)

Consecutive frame (...)

Node B

Vincent Dupont (OTA keys)

RIOT and CAN

RIOT Summit 2017
1. What is CAN?

2. CAN in RIOT
   - Overall architecture
   - candev
   - Link layer
   - conn_can

3. Use case example: OBD

4. Future
Architecture

- conn_can
- conn_can_isotp
- isotp
- CAN link layer
  - raw / router / pkt / dll
  - candev
- drivers

Vincent Dupont (OTA keys)
RIOT and CAN
RIOT Summit 2017
Architecture

- conn_can
- conn_can_isotp
- isotp
- CAN link layer
  - raw / router / pkt / dll
  - candev
- drivers

Implement candev

Vincent Dupont (OTA keys)  RIOT and CAN  RIOT Summit 2017
Architecture

conn_can

conn_can_isotp

isotp

CAN link layer

raw / router / pkt / dll

candev

drivers

conn_can user interfaces

implement candev
Architecture

- conn_can user interfaces
- CAN stack: isotp, link layer
- implement candev
Architecture

- **conn_can**
- **conn_can_isotp**
  - **isotp**
  - **CAN link layer**
    - **raw / router / pkt / dll**
    - **candev**
  - **drivers**

**conn_can user interfaces**

**CAN stack: isotp, link layer**

1 thread per layer

**implement candev**
typedef struct candev candev_t;

typedef void (*candev_event_cb_t)(candev_t *dev, candev_event_t event, void *arg);

struct candev {
    const struct candev_driver *driver;
    candev_event_cb_t event_callback;
    void *isr_arg;
    /* CAN specific */
    struct can_bittiming bittiming;
    enum can_state state;
};
Candev driver interface

typedef struct candev_driver {
    int (*init)(candev_t *dev);
    void (*isr)(candev_t *dev);

    /* ... */
}
candev_driver_t;
typedef struct candev_driver {
    int (*init)(candev_t *dev);
    void (*isr)(candev_t *dev);
    int (*send)(candev_t *dev,
                const struct can_frame *frame);
    int (*abort)(candev_t *dev,
                 const struct can_frame *frame);
    /* ... */
} candev_driver_t;
Candev driver interface

typedef struct candev_driver {
    int (*init)(candev_t *dev);
    void (*isr)(candev_t *dev);
    int (*send)(candev_t *dev,
                const struct can_frame *frame);
    int (*abort)(candev_t *dev,
                const struct can_frame *frame);
    int (*get)(candev_t *dev, canopt_t opt,
               void *value, size_t max_len);
    int (*set)(candev_t *dev, canopt_t opt,
               void *value, size_t value_len);

    /* ... */
}

} candev_driver_t;
typedef struct candev_driver {
    int (*init)(candev_t *dev);
    void (*isr)(candev_t *dev);
    int (*send)(candev_t *dev,
                const struct can_frame *frame);
    int (*abort)(candev_t *dev,
                 const struct can_frame *frame);
    int (*get)(candev_t *dev, canopt_t opt,
               void *value, size_t max_len);
    int (*set)(candev_t *dev, canopt_t opt,
               void *value, size_t value_len);
    int (*set_filter)(candev_t *dev,
                      const struct can_filter *filter);
    int (*remove_filter)(candev_t *dev,
                         const struct can_filter *filter);
} candev_driver_t;
Link Layer

- raw_can (dll.c)
- router (router.c)
- pkt (pkt.c): wraps
- can_dll (dll.c)
- candev (device.c)
Link Layer: sending

User \[\text{raw\_can} \rightarrow \text{can\_dll} \rightarrow \text{can\_dev}\]

raw\_can\_send \[\rightarrow \text{msg\_send} \rightarrow \text{msg\_send} \leftarrow \text{ok} \leftarrow \text{can\_dll\_dispatch\_tx\_conf}\]
Link Layer: receiving, step 1: register filter

User x \rightarrow raw\_can

raw\_can\_subscribe\_rx

can\_router\_register

already registered?

←

Register filter only if needed

set filter

ok / ko

ok / ko

router \rightarrow can\_dll \rightarrow candev
Link Layer: receiving, step 2: receive

User x ➔ raw_can ➔ router ➔ can_dll ➔ candev

- `can_dll_dispatch_rx_ind`
- `can_router_dispatch_rx_ind`

Dispatch to user threads

- 1 msg per thread
- `msg_send`
- `inc usage counter`
- `ok / ko`

Vincent Dupont (OTA keys)
RIOT and CAN
RIOT Summit 2017
Link Layer: receiving, step 3: free

Free each user thread

raw_can_free_frame

can_router_free_frame

decrement usage counter

if counter == 0

free frame
- Provide synchronous calls to interact with CAN stack
- “connection-oriented” interface
- More suitable for user code and complex applications
conn_can

- Provide synchronous calls to interact with CAN stack
- “connection-oriented” interface
- More suitable for user code and complex applications

Available functions:
- conn_can_raw_create: needed only to receive, set filters
- conn_can_raw_send
- conn_can_raw_recv
- conn_can_raw_close: unset filters
Available functions:

- `conn_can_isotp_create`
- `conn_can_isotp_bind`: set filter
- `conn_can_isotp_send`
- `conn_can_isotp_recv`
- `conn_can_isotp_close`: unset filter

Bonus:

- `conn_can_isotp_select`: if module `CONN_CAN_ISOTP_MULTI` used, a thread can bind multiple isotp connections
1. What is CAN?

2. CAN in RIOT

3. Use case example: OBD

4. Future
OBD

off-board

Tester

on-board

OBD port

ECU 1

ECU 2

ECU 3..8
On top of CAN and ISO-TP
Up to 8 ECUs
On top of CAN and ISO-TP

Up to 8 ECUs

<table>
<thead>
<tr>
<th>ECU</th>
<th>Tester Address</th>
<th>ECU Address</th>
</tr>
</thead>
<tbody>
<tr>
<td>Broadcast</td>
<td>0x7DF</td>
<td></td>
</tr>
<tr>
<td>#0</td>
<td>0x7E0</td>
<td>0x7E8</td>
</tr>
<tr>
<td>#n</td>
<td>0x7En</td>
<td>0x7En + 8</td>
</tr>
</tbody>
</table>
### OBD: request example

<table>
<thead>
<tr>
<th>ID</th>
<th>Request</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x7DF</td>
<td>0x02 0x01 0x0D</td>
</tr>
<tr>
<td>Broadcast</td>
<td>isotp mode pid</td>
</tr>
<tr>
<td></td>
<td>SF 1 Vehicle Speed</td>
</tr>
</tbody>
</table>

- **isotp**: isochronous mode
- **SF**: service frame
- **1 byte read**: vehicle
- **3 bytes resp**: speed
### OBD: request example

<table>
<thead>
<tr>
<th>ID</th>
<th>Request</th>
</tr>
</thead>
</table>
| 0x7DF   | 0x02 0x01 0x0D 0x55 0x55 0x55 0x55 0x55 | Broadcast  
|         | isotp  
|         | mode  
|         | 1     
|         | read  
|         | pid   
|         | Vehicle Speed             | padding |

<table>
<thead>
<tr>
<th>ID</th>
<th>Response</th>
</tr>
</thead>
</table>
| 0x7E8   | 0x03 0x41 0x0D 0x7F 0x55 0x55 0x55 0x55 | ECU #1   
|         | isotp  
|         | mode  
|         | 1     
|         | resp  
|         | pid   
|         | Vehicle Speed              | value  
|         | 127 km/h                   | padding |
9 modes

PID (Parameter Identifiers)
- 9 modes
- PIDs (Parameter Identifiers)

### Example

<table>
<thead>
<tr>
<th>Mode</th>
<th>PID</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0x0</td>
<td>PIDs supported (range 0x01 – 0x20)</td>
</tr>
<tr>
<td>1</td>
<td>0xC</td>
<td>Engine RPM</td>
</tr>
<tr>
<td>1</td>
<td>0xD</td>
<td>Vehicle speed</td>
</tr>
<tr>
<td>9</td>
<td>0x2</td>
<td>VIN (Vehicle Identification Number)</td>
</tr>
<tr>
<td>3</td>
<td>No PID</td>
<td>Diagnostic Trouble Codes (DTCs)</td>
</tr>
</tbody>
</table>
/* Step 1: Prepare to receive ECUs responses */

conn_can_raw_t raw_conn;
int ifnum = 0;
struct can_filter filter = {
    .can_id = 0x7E8,
    .can_mask = 0xffffffff8,
};
conn_can_raw_create(&raw_conn, &filter, 1, ifnum, 0);
/* Step 1: Prepare to receive ECUs responses */
conn_can_raw_t raw_conn;
int ifnum = 0;
struct can_filter filter = {
    .can_id = 0x7E8,
    .can_mask = 0xffffffff8,
};
conn_can_raw_create(&raw_conn, &filter, 1, ifnum, 0);

/* Step 2: Send request */
struct can_frame frame;
memset(frame.data, 0x55, 8); // init with padding
frame.data[0] = 0x02; // isotp header: SF, l=2
frame.data[1] = 0x1; // mode
frame.data[2] = 0x0; // pid (supported PIDs range 0 - 0x20)
frame.can_dlc = 8; // Frame length
frame.can_id = 0x7DF; // Broadcast ID
conn_can_raw_send(&raw_conn, &frame, 0);
OBD: a bit of code

/* Step 3: Wait for ECUs, save ECU address supporting PID */
canid_t ecus[8];
int nb_ecus = 0;
while (conn_can_raw_recv(&raw_conn, &frame, TIMEOUT) > 0) {
    if (check_frame(&frame)) {
        ecus[nb_ecus++] = frame.can_id;
    }
}

Vincent Dupont (OTA keys)
RIOT and CAN
RIOT Summit 2017
/* Step 4: Send actual request */
/* For each ECU supporting PID */
canid_t ecu_addr = ecus[i];
canid_t tester_addr = ecu_addr - 8;
/* Init ISO-TP with addresses and padding */
struct isotp_options options = {
    .tx_id = tester_addr, .rx_id = ecu_addr,
    .txpad_content = 0x55, .flags = CAN_ISOTP_TX_PADDING,
};
/* Create and bind connection */
conn_can_isotp_create(&isotp_conn, &options, ifnum);
conn_can_isotp_bind(&isotp_conn);
/* Step 4: Send actual request */
/* For each ECU supporting PID */
canid_t ecu_addr = ecus[i];
canid_t tester_addr = ecu_addr - 8;
/* Init ISO-TP with addresses and padding */
struct isotp_options options = {
    .tx_id = tester_addr,  .rx_id = ecu_addr,
    .txpad_content = 0x55, .flags = CAN_ISOTP_TX_PADDING,
};
/* Create and bind connection */
conn_can_isotp_create(&isotp_conn, &options, ifnum);
conn_can_isotp_bind(&isotp_conn);
/* Send request frame (as in previous slide) */
conn_can_raw_send(&raw_conn, &frame, 0);
/* Step 4: Send actual request */
/* For each ECU supporting PID */
canid_t ecu_addr = ecus[i];
canid_t tester_addr = ecu_addr - 8;
/* Init ISO-TP with addresses and padding */
struct isotp_options options = {
    .tx_id = tester_addr, .rx_id = ecu_addr,
    .txpad_content = 0x55, .flags = CAN_ISOTP_TX_PADDING,
};
/* Create and bind connection */
conn_can_isotp_create(&isotp_conn, &options, ifnum);
conn_can_isotp_bind(&isotp_conn);
/* Send request frame (as in previous slide) */
conn_can_raw_send(&raw_conn, &frame, 0);
/* Wait for response */
uint8_t buf[32];
conn_can_isotp_recv(&isotp_conn, buf, sizeof(buf), TIMEOUT);
conn_can_isotp_close(&isotp_conn);
/* Buf contains ECU response */
/* If multi frame, isotp layer reconstructed it */
1. What is CAN?

2. CAN in RIOT

3. Use case example: OBD

4. Future
What for the future?

CAN stack:
- Add actual drivers for hardware
- Merge candev and netdev
- Re-use parts of gnrc for CAN stack
- New higher-layer protocols (OBD, Broadcast Manager, J1939, CANopen, DeviceNet ...)
- Adapt stack to CAN-FD
What for the future?

CAN stack:
- Add actual drivers for hardware
- Merge candev and netdev
- Re-use parts of gnrc for CAN stack
- New higher-layer protocols (OBD, Broadcast Manager, J1939, CANopen, DeviceNet ...)
- Adapt stack to CAN-FD

RIOT and vehicle:
- DoIP (Diagnostic over IP): the future of vehicle diagnostics
- Connected car
Conclusion

**RIOT**:

- fun and easy to hack and use, thanks to its community
- now the only free OS for small embedded systems with a CAN stack with full ISO-TP support
Thank you!
Other physical layers

- High-Speed CAN: ISO 11898-2, most commonly used
- Low-Speed (aka Fault-Tolerant) CAN: ISO 11898-3. Up to 125kbit/s
- Low-Speed high-voltage: ISO 11992-1. For truck-trailer point-to-point communication
- Single-Wire CAN: SAE J2411

Source: https://www.can-cia.org/can-knowledge/can/systemdesign-can-physicallayer/
Link layer: errors

5 sources of error:
- Bit monitoring
- Bit stuffing
- Frame check
- Acknowledgement check
- CRC check

Error frame

6 consecutives dominant or recessive bits

2 error counters (Tx and Rx):
- $c \leq 127$: error active
- $127 < c \leq 255$: error passive
- $c > 255$: bus off
Bit timing

Bit divided in time quanta (TQ) \((1TQ = \frac{Clock}{BRP})\), 4 segments:

- Synchronization (SYNC)
- Propagation (PROP)
- Phase segment 1 (PS1)
- Phase segment 2 (PS2)

Sample point between PS1 and PS2
### Header example:

<table>
<thead>
<tr>
<th>Value (hex)</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>05</td>
<td>SF, length 5 bytes</td>
</tr>
<tr>
<td>10 08</td>
<td>FF, length 8 bytes</td>
</tr>
<tr>
<td>1F FF</td>
<td>FF, length 4095 bytes</td>
</tr>
<tr>
<td>21</td>
<td>CF, index 1</td>
</tr>
<tr>
<td>25</td>
<td>CF, index 5</td>
</tr>
<tr>
<td>30</td>
<td>FC CTS (clear to send)</td>
</tr>
<tr>
<td>31</td>
<td>FC Wait</td>
</tr>
<tr>
<td>32</td>
<td>FC Overflow</td>
</tr>
</tbody>
</table>
Transport layer: ISO-TP

UDS (Unified Diagnostic Services): ISO 14229-1
  - Services
  - Tester sends a request, ECU responds
  - Applications: diagnostic (read/write data, error codes), firmware update (read/write memory), etc.

OBD (On-Board Diagnostic): ISO 15031