RIOT and CAN

Vincent Dupont

OTA keys

RIOT Summit
September 25-26, 2017

OTAkeys

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)

OTA keys

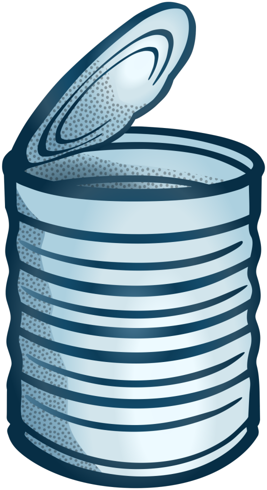
Goal of this presentation

1. CAN bus technology
2. RIOT CAN stack
3. CAN stack usage example

Content

1. What is CAN?
   - Physical Layer
   - Link layer
   - ISO-TP
2. CAN in RIOT
3. Use case example: OBD
4. Future

What is CAN?



Physical Layer: ISO 11898-2

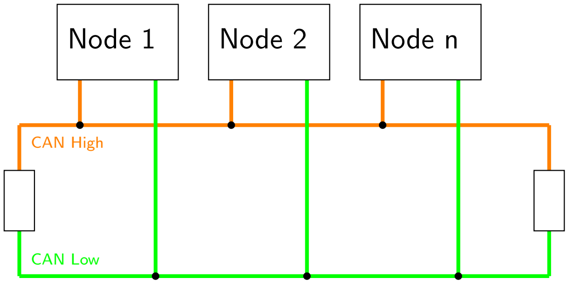
Definition

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

Physical Layer: ISO 11898-2

Definition

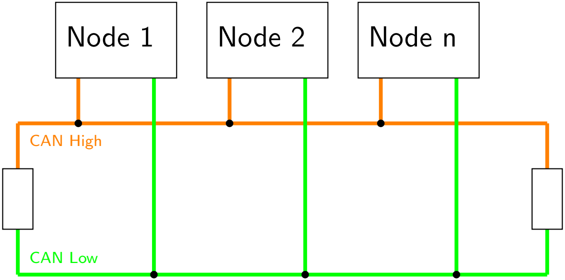
CAN is a multi-master serial bus standard for connecting Electronic Control Units [ECUs] also known as nodes.



Physical Layer: ISO 11898-2

Definition

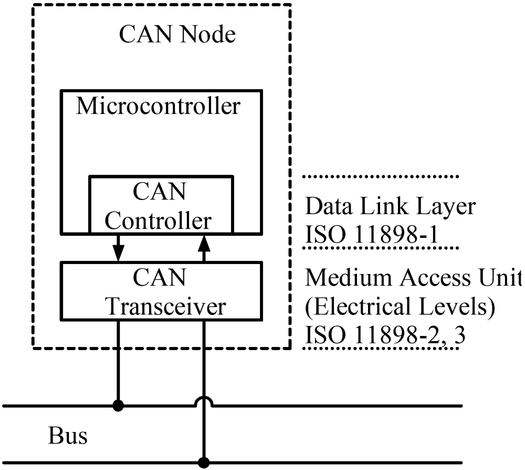
CAN is a multi-master serial bus standard for connecting Electronic Control Units [ECUs] also known as nodes.



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)

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 A	079	0 (SOF)	0	0	0	0	1	1	1	1	0	0	1
Node B	080	0	0	0	0	1	Stops transmitting						
Node C	700	0	1	Stops transmitting									

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) $\rightarrow$ arbitration phase:

Node A	079	0 (SOF)	0	0	0	0	1	1	1	1	0	0	1
Node B	080	0	0	0	0	1	Stops transmitting						
Node C	700	0	1	Stops transmitting									

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 A	079	0 (SOF)	0	0	0	0	1	1	1	1	0	0	1
Node B	080	0	0	0	0	1	Stops transmitting						
Node C	700	0	1	Stops transmitting									

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) $\rightarrow$ arbitration phase:

Node A	079	0 (SOF)	0	0	0	0	1	1	1	1	0	0	1
Node B	080	0	0	0	0	1	Stops transmitting						
Node C	700	0	1	Stops transmitting									

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) $\rightarrow$ arbitration phase:

Node A	079	0 (SOF)	0	0	0	0	1	1	1	1	0	0	1
Node B	080	0	0	0	0	1	Stops transmitting						
Node C	700	0	1	Stops transmitting									

- Filters: receive if $(can\_id \& mask) == filter$

ISO-TP: ISO 15765-2

- Segmentation
- Up to 4095 bytes
- Use a pair of CAN IDs
- “Channel” between 2 nodes

ISO-TP: ISO 15765-2

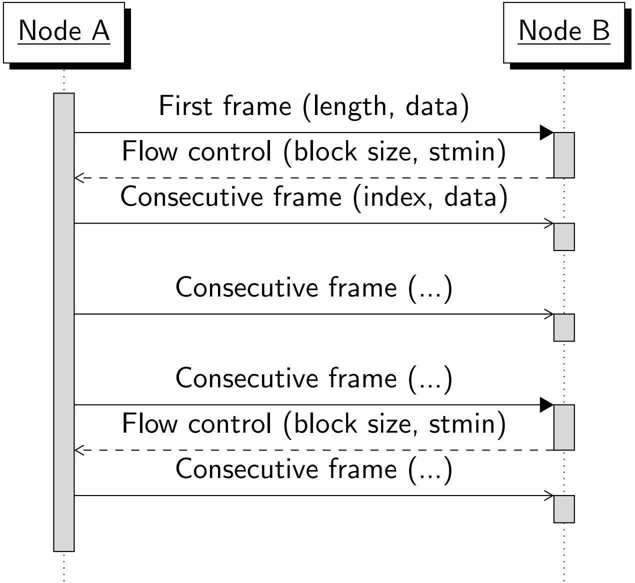
- Segmentation
- Up to 4095 bytes
- Use a pair of CAN IDs
- “Channel” between 2 nodes

Header (4 bits):

Value	Definition
0	Single frame (SF)
1	First frame (FF)
2	Consecutive frame (CF)
3	Flow control (FC)

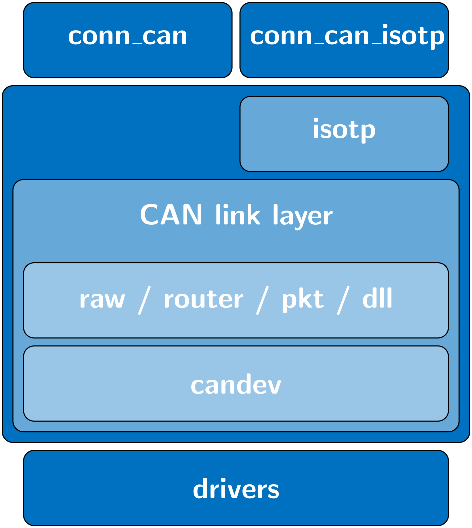
OTA keys

ISO-TP: ISO 15765-2, Example

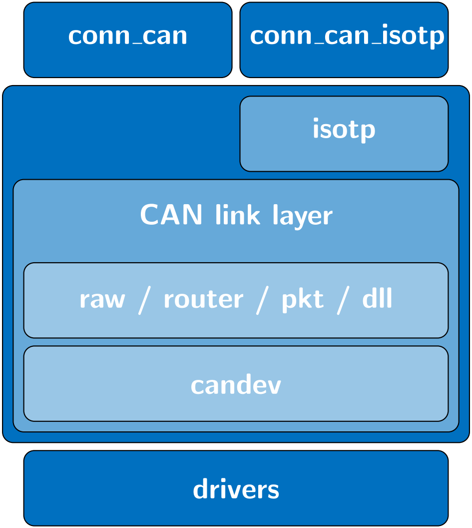


Content

Architecture

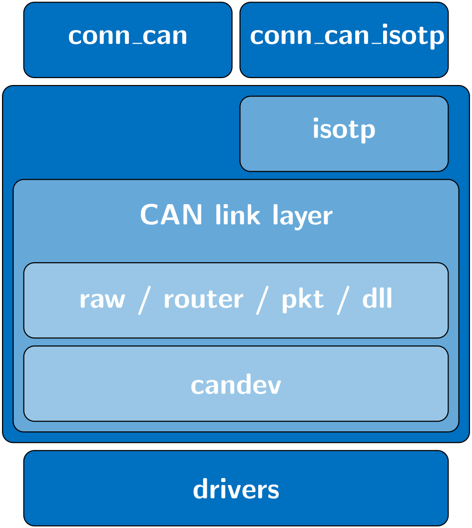


Architecture



implement candev

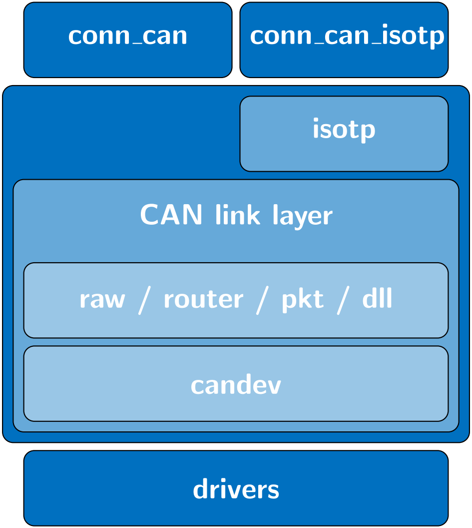
Architecture



conn_can user interfaces

implement candev

Architecture

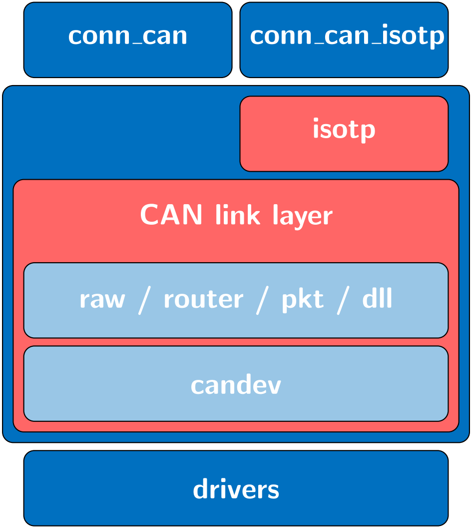


conn_can user interfaces

CAN stack: isotp, link layer

implement candev

Architecture



conn_can user interfaces

CAN stack: isotp, link layer

1 thread per layer

implement candev

Candev structure

```c
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;
```

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);

    /* ... */

} candev_driver_t;
```

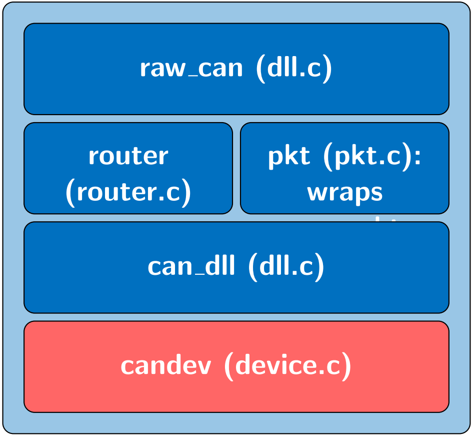
Candev driver interface

```
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;
```

Candev driver interface

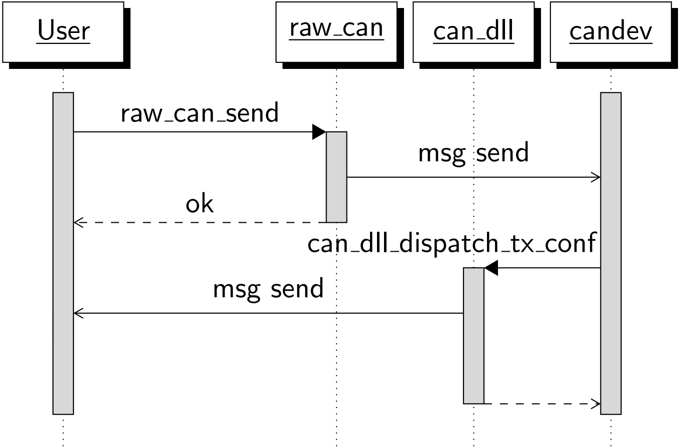
```
typedef struct candev_driver {
    int (*init)(candev_t *dev);
    void (*isr)(candev_t *dev);
    int (*send)(candev_t *dev,
                const struct can_frame *frame);
    int (*abort)(candev_t *dev,
                 const struct can_frame *frame);
    int (*get)(candev_t *dev, canopt_t opt,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

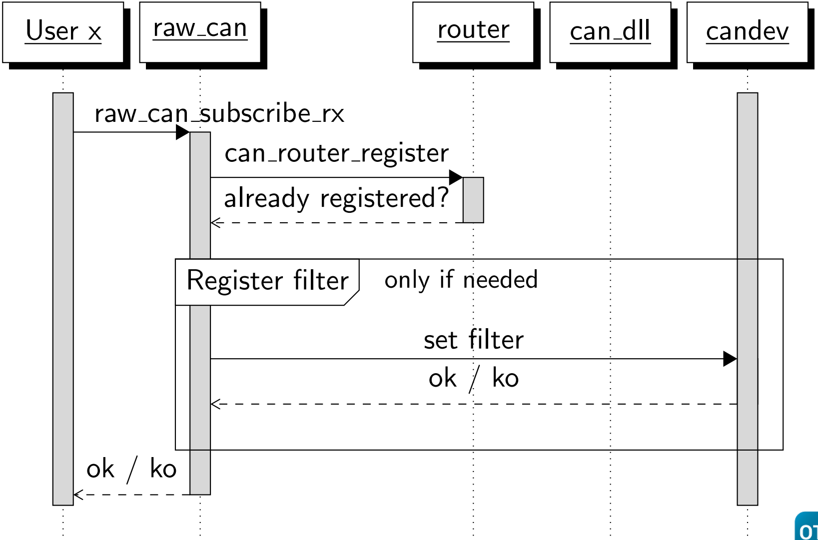


raw_can (dll.c)
router (router.c)
pkt (pkt.c): wraps
can_dll (dll.c)
candev (device.c)

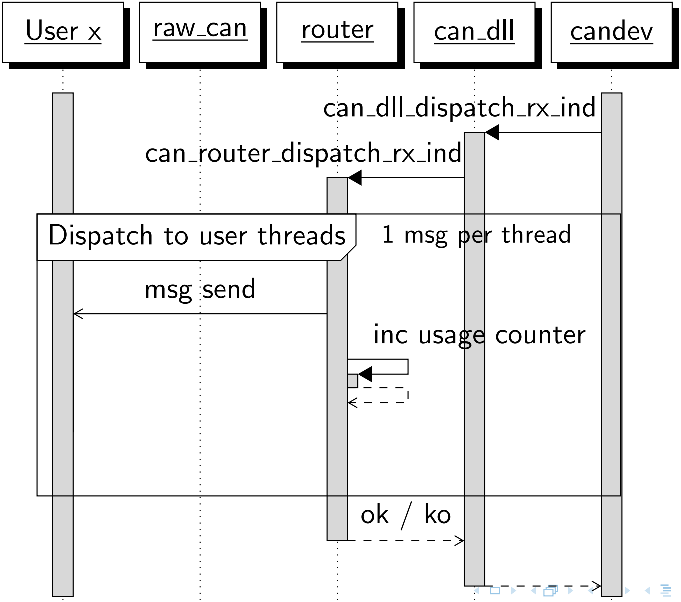
Link Layer: sending



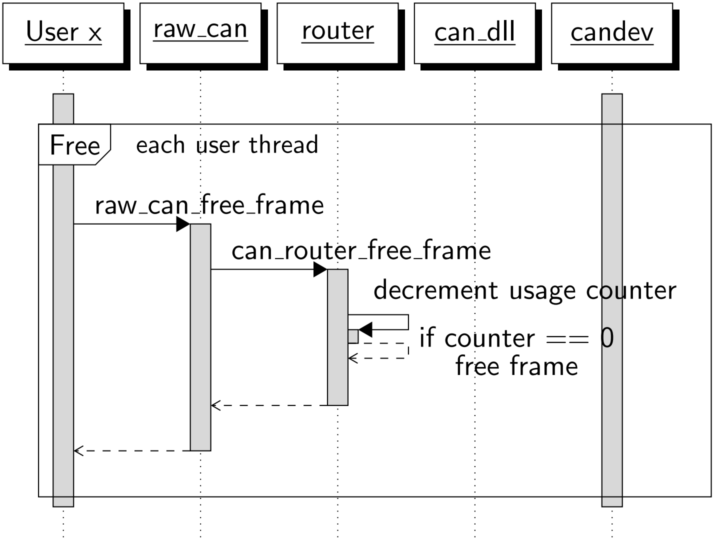
Link Layer: receiving, step 1: register filter



Link Layer: receiving, step 2: receive



Link Layer: receiving, step 3: free



conn_can

- 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

conn_can_isotp

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

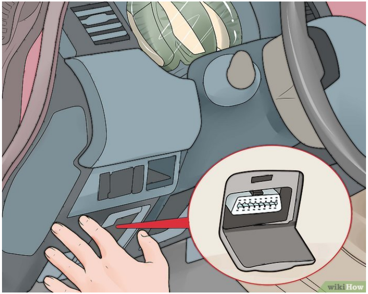
Bonus:

- `conn_can_isotp_select`: if module `CONN_CAN_ISOTP_MULTI` used, a thread can bind multiple isotp connections

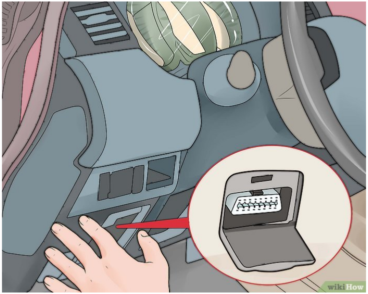
Content

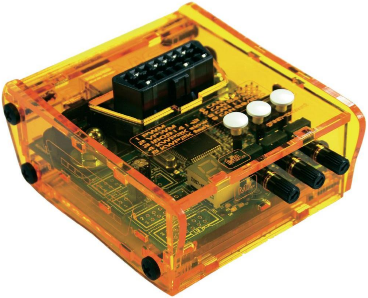
OTA keys

OBD

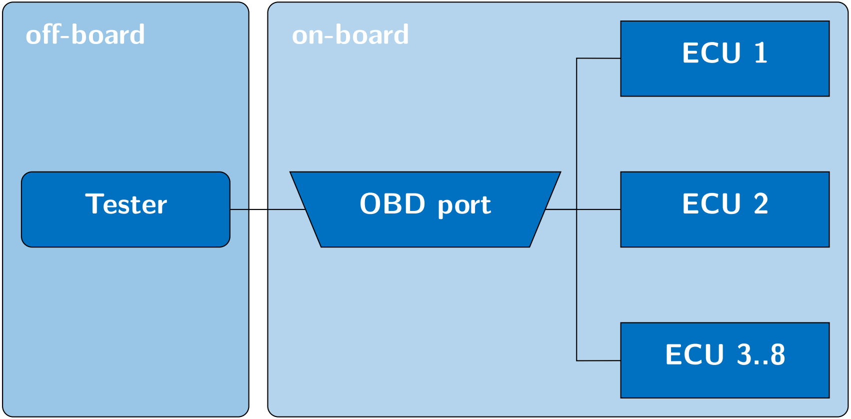


OBD





OBD



OBD

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

OBD

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

ECU	Tester Address	ECU Address
Broadcast	0x7DF	
#0	0x7E0	0x7E8
#n	0x7En	0x7En + 8

OBD: request example

ID	Request							
0x7DF	0x02	0x01	0x0D	0x55	0x55	0x55	0x55	0x55
Broadcast	isotp SF 2bytes	mode 1 read	pid Vehicle Speed	padding				

OBD: request example

ID	Request							
0x7DF	0x02	0x01	0x0D	0x55	0x55	0x55	0x55	0x55
Broadcast	isotp SF 2bytes	mode 1 read	pid Vehicle Speed	padding				

ID	Response							
0x7E8	0x03	0x41	0x0D	0x7F	0x55	0x55	0x55	0x55
ECU #1	isotp SF 3bytes	mode 1 resp	pid Vehicle Speed	value 127 km/h	padding			

OBD

- 9 modes
- PIDs (Parameter Identifiers)

OBD

- 9 modes
- PIDs (Parameter Identifiers)

Example

Mode	PID	Description
1	0x0	PIDs supported (range 0x01 – 0x20)
1	0xC	Engine RPM
1	0xD	Vehicle speed
9	0x2	VIN (Vehicle Identification Number)
3	No PID	Diagnostic Trouble Codes (DTCs)

OBD: a bit of code

```
/* 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 = 0xfffffff8,
};
conn_can_raw_create(&raw_conn, &filter, 1, ifnum, 0);
```

OBD: a bit of code

```
/* 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 = 0xfffffff8,
};
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;
    }
}
```

OBD: a bit of code

```
/* 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);
```

OBD: a bit of code

```c
/* 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);
```

OBD: a bit of code

```
/* 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 */
```

Content

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/`

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

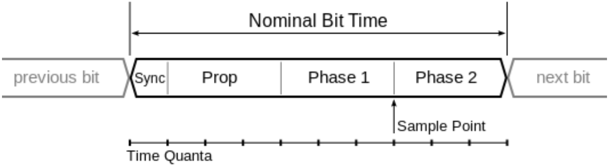
Bit timing

Bit timing

Bit divided in time quanta (TQ) ($1TQ = Clock/BRP$), 4 segments:

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

Sample point between PS1 and PS2



ISO-TP: ISO 15765-2, Header

Header example:

Value (hex)	Definition
05	SF, length 5 bytes
10 08	FF, length 8 bytes
1F FF	FF, length 4095 bytes
21	CF, index 1
25	CF, index 5
30	FC CTS (clear to send)
31	FC Wait
32	FC Overflow

Diagnostic

- 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