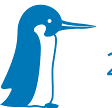
Linux-CAN Subsystem BoF
State of the Union and Outlook

Marc Kleine-Budde – mkl@pengutronix.de
Oleksij Rempel – ore@pengutronix.de

https://www.pengutronix.de

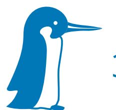
CAN – the basics

- broadcast bus
- no flow control
- small frames
  - CAN-CC: 11/29 bit address, 0…8 bytes data
  - CAN-FD: 11/29 bit address, 0…64 bytes data
  - CAN-XL: more elaborated addressing, 0…2048 bytes data (HW not yet available)



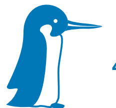
RX-Path – the usual approach (without DMA)

- IP core
  - receive packet, generate IRQ
- IRQ handler
  - mask RX-IRQ
  - schedule NAPI
- NAPI (soft-IRQ or kernel thread)
  - allocate skb
  - copy data from IP core to skb
  - push skb to networking stack → `netif_receive_skb()`



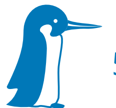
RX-Path – the problem with CAN

- IP cores usually don't support DMA
- limited amount of internal buffers (32 frames)
- high number of CAN frames/s, up to 10k/s
- too high latency between IRQ handler and NAPI causes packet loss



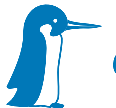
RX-Path – the good old days before NAPI?

- IRQ handler
  - allocate skb
  - copy data from IP core to skb
  - push skb to networking stack → `netif_rx()`
  - `netif_rx()`
    - works from IRQ
    - `netif_receive_skb()` doesn't
    - prone to Out-of-Order reception – bad for CAN



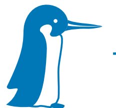
RX-Path – solution: rx-offload - IRQ

- IRQ handler
  - allocate skb
  - copy data from IP core to skb
  - add skb to rx-offload queue:
    `can_rx_offload_queue_tail()`,
    `can_rx_offload_queue_timestamp()`
  - trigger rx-offload-NAPI:
    `can_rx_offload_irq_finish()`,
    `can_rx_offload_threaded_irq_finish()`



RX-Path – solution: rx-offload - NAPI

- NAPI
  - iterate over queue
  - push skb to networking stack → `netif_receive_skb()`

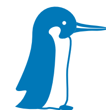


RX/TX timestamping

- convert from controller's internal clock to kernel's representation (in nanoseconds)
- don't re-invent the wheel, use cyclecounter/timecounter
-

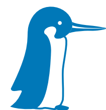
```
struct cyclecounter {
    u64 (*read)(const struct cyclecounter *cc);
    u64 mask;
    u32 mult;
    u32 shift;
};
// ns = ((read() & mask) * mult) >> shift;
```

- `struct timecounter tc;`



What is J1939, and why is it important?

- SAE J1939 is a **vehicle bus standard**. (Similar to TCP/IP in networking, it manages communication between vehicle components.)
- Provides standardized communication and diagnostic functionalities, allowing integration of components from **different vendors**.
- Widely adopted in **automotive**, **agricultural**, and **marine** industries.
- Provides **transport protocols** for larger payloads (like TCP for data transmission).
- Includes **address claiming** mechanisms (similar to DHCP in networking).
- Defines **application-specific standards** for various use cases (comparable to application layer protocols like HTTP or FTP).



J1939 stack in the wild

Fun and experimental projects

Connecting Euro Truck Simulator 2 to a real MAN TGX instrument cluster by using J1939 kernel stack

Open-source tools in can-utils

- **j19393acd**: Address Claim daemon (similar to DHCP).
- **j1939cat**: Streams data over CAN using the J1939 transport protocol.
- **j1939spy**: Monitors CAN bus traffic.
- **j1939sr**: Simple send/receive utility.
- **isobusfs**: Implements ISO 11783 file server/client. (Similar to FTP)
- **j1939-timedate**: Requests and syncs time/date over CAN. (Similar to one shot NTP)

Ongoing work - J1939 vehicle position service

GPS receiver

GPSD

GPSD to J1939

CAN bus

J1939 to GPSD

GPSD

App

App

J1939 vehicle position service

TODO

Ongoing work - J1939 vehicle position service

GPSD

```
Time:          2024-07-27T15:00:59.000Z (18)
Latitude:           51.88891970 N
Longitude:          10.06607080 E
Alt (HAE, MSL):      678.898,     526.601 ft
Speed:               0.01 mph
Track (true, var):       2.3,    3.1      deg
Climb:               0.59 ft/min
Status:           3D FIX (11 secs)
```

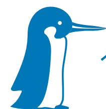
```
                 Seen 26/Used 15
GNSS    PRN  Elev   Azim   SNR Use
GP  3     3  17.0  117.0  16.0  Y
GP  4     4  47.0   67.0  27.0  Y
GP  6     6  53.0  229.0  28.0  Y
GP  7     7  39.0  173.0  27.0  Y
GP  9     9  87.0   60.0  32.0  Y
GP 11    11  42.0  296.0  40.0  Y
GP 20    20  21.0  301.0  42.0  Y
GP 26    26  12.0   38.0  37.0  Y
```

CAN dump

```
(170.851699)  vcan0  18FEF340   [8]  BB 13 19 9C A9 6A 2B 83
(005.005778)  vcan0  18FEF340   [8]  BB 13 19 9C A9 6A 2B 83
(005.005617)  vcan0  18FEF340   [8]  C0 13 19 9C A8 6A 2B 83
(005.005793)  vcan0  18FEF340   [8]  C0 13 19 9C A8 6A 2B 83
(005.005495)  vcan0  18FEF340   [8]  C6 13 19 9C A8 6A 2B 83
(005.005875)  vcan0  18FEF340   [8]  C6 13 19 9C A8 6A 2B 83
(005.005508)  vcan0  18FEF340   [8]  CB 13 19 9C A6 6A 2B 83
(005.002889)  vcan0  18FEF340   [8]  CB 13 19 9C A6 6A 2B 83
```

Client

```
Latitude: 51.8889158, Longitude: 10.0660648
Latitude: 51.8889163, Longitude: 10.0660646
Latitude: 51.8889163, Longitude: 10.0660646
Latitude: 51.8889163, Longitude: 10.0660646
Latitude: 51.8889163, Longitude: 10.0660646
Latitude: 51.8889163, Longitude: 10.0660646
Latitude: 51.8889163, Longitude: 10.0660646
```

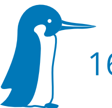


Validation and Testing

- Proprietary Tools:
  - Used by vendors for field validation of J1939 applications.
  - Ensures reliability in real-world environments.
- Open-Source Testing:
  - Custom scripts in the **can-tests** repository created to reproduce and fix bugs.
  - Tests validate the stack using tools like j1939cat and j19393acd.
- Community Feedback:
  - Contributions from companies like Huawei and Protonic (Protocol-specific validation), Google (syzkaller) helped to improve the stack.

Community Involvement

- Building a Community:
  - Encourage hackers and developers to explore and contribute to the J1939 stack.
  - Examples include Raspberry Pi projects and DIY applications using the stack.
- Vendor Sponsorship:
  - Potential for vendors to sponsor further development and upstream work.
  - Sponsorship can drive new features and enhancements.
- Get Involved:
  - Contribute code, report issues, or suggest features via the can-utils GitHub repository.
  - Collaboration opportunities for both hobbyists and industry professionals.



Related repositories

- Linux kernel
  - https://www.kernel.org/
- CAN utils
  - https://github.com/linux-can/can-utils
- CAN tests
  - https://github.com/linux-can/can-tests