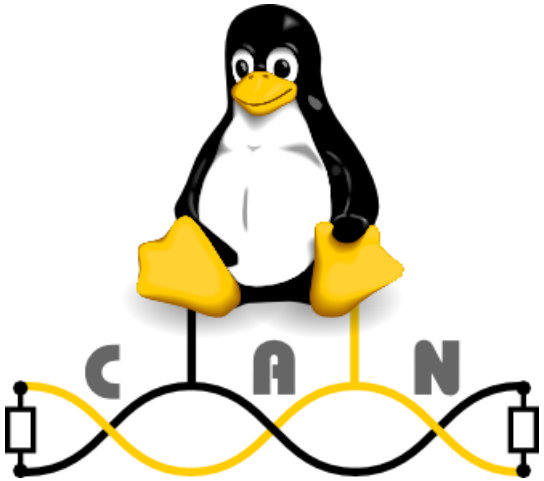
VOLKSWAGEN

AKTIENGESELLSCHAFT



Linux and ISO 15765-2 with CAN FD

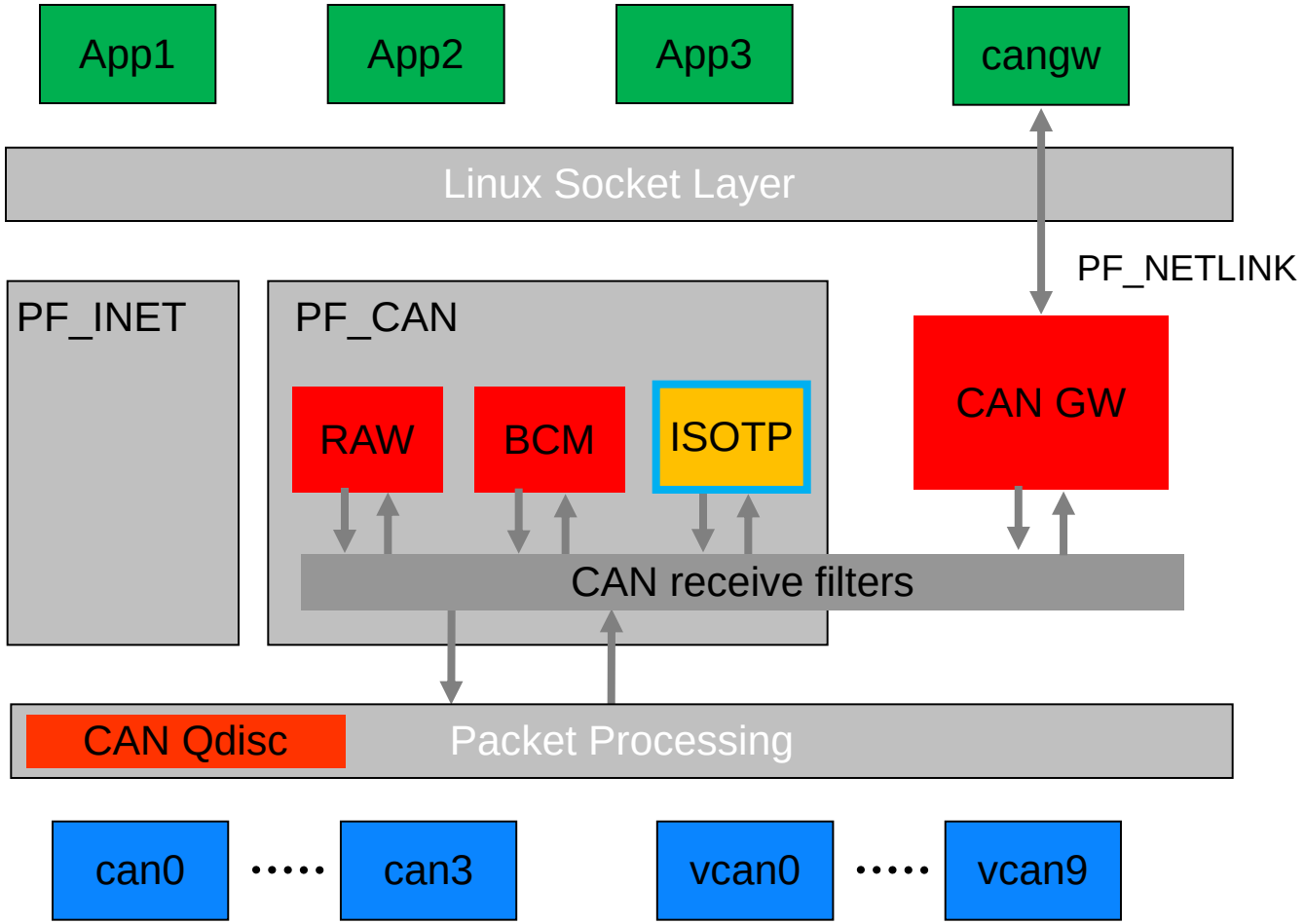
15th international CAN Conference 2015

Dr. Oliver Hartkopp (Volkswagen AG)

Content

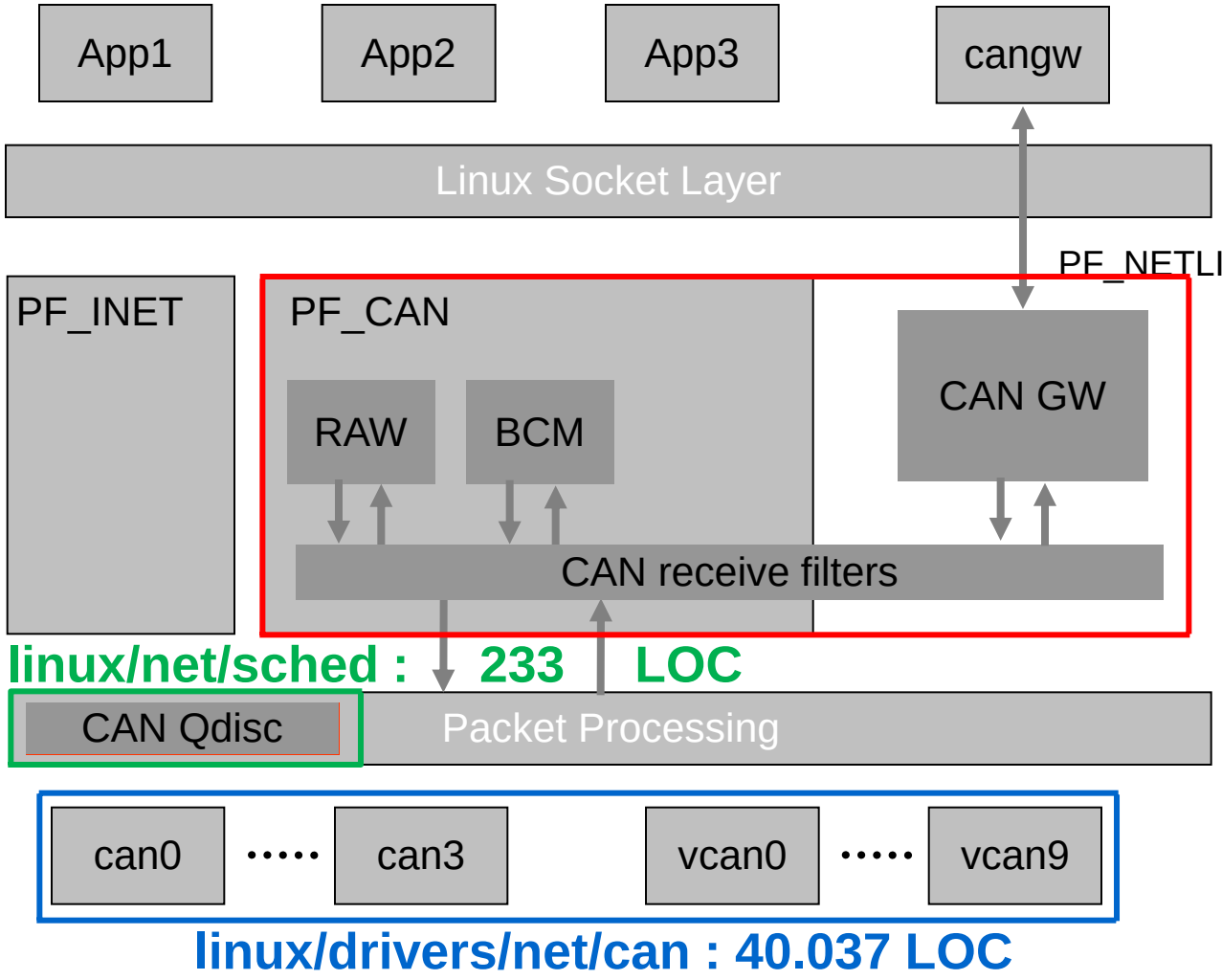
1. CAN Protocols inside the Linux network layer
2. CAN FD Integration & Status
3. Transport protocol ISO 15765-2:2015 with CAN FD
4. CAN Hardware

What's inside Linux CAN?



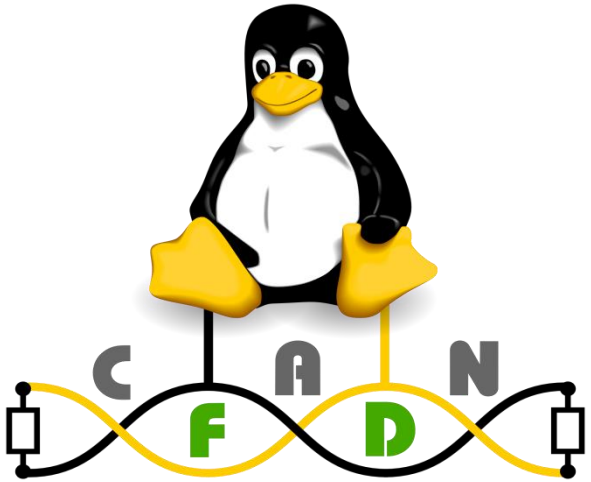
ISOBUS/J1939 Reminder

Lines of Code summary (Linux 4.3)



linux/net/can : 5.227 LOC

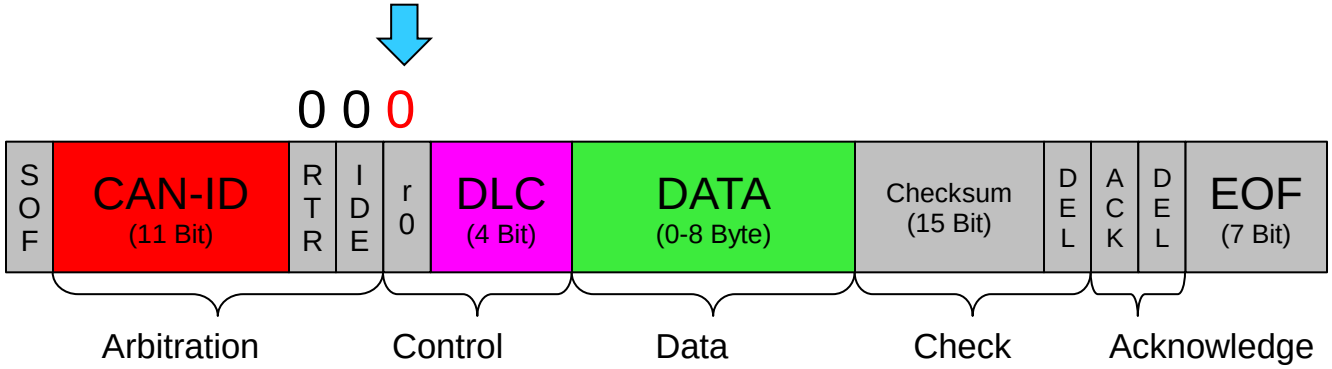
```
at91_can mscan slcan
c_can bfin_can d_can
sja1000 vcan flexcan
ti_hecc xilinx_can
cc770 mcp251x m_can
rcar_can softing
peak_usb usb8dev
ems_usb kvaser_usb
esd_usb2 gs_usb
janz_ican3 pch_can
```

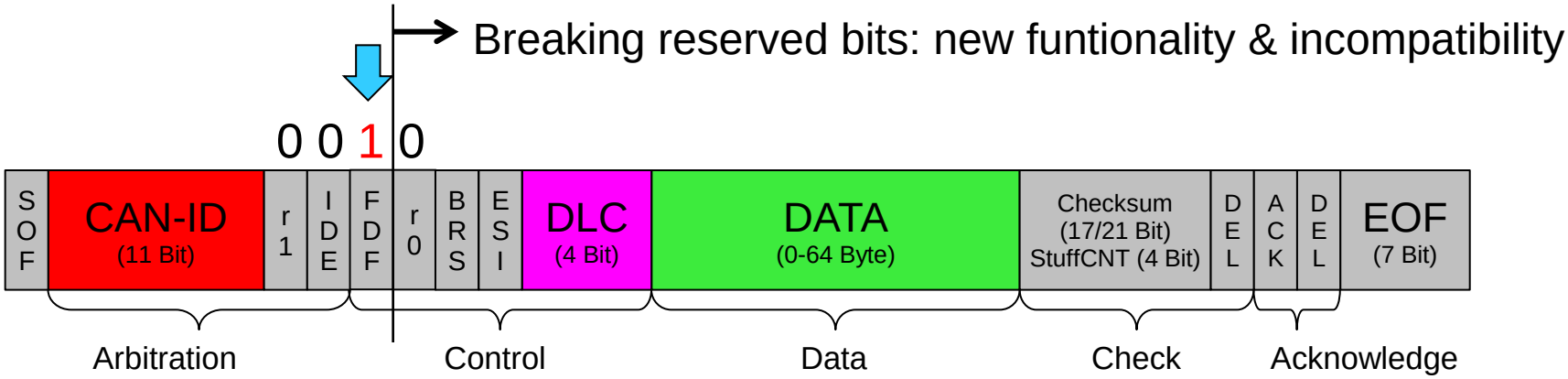


CAN FD Integration in Linux
Adopting CAN with Flexible Data rate

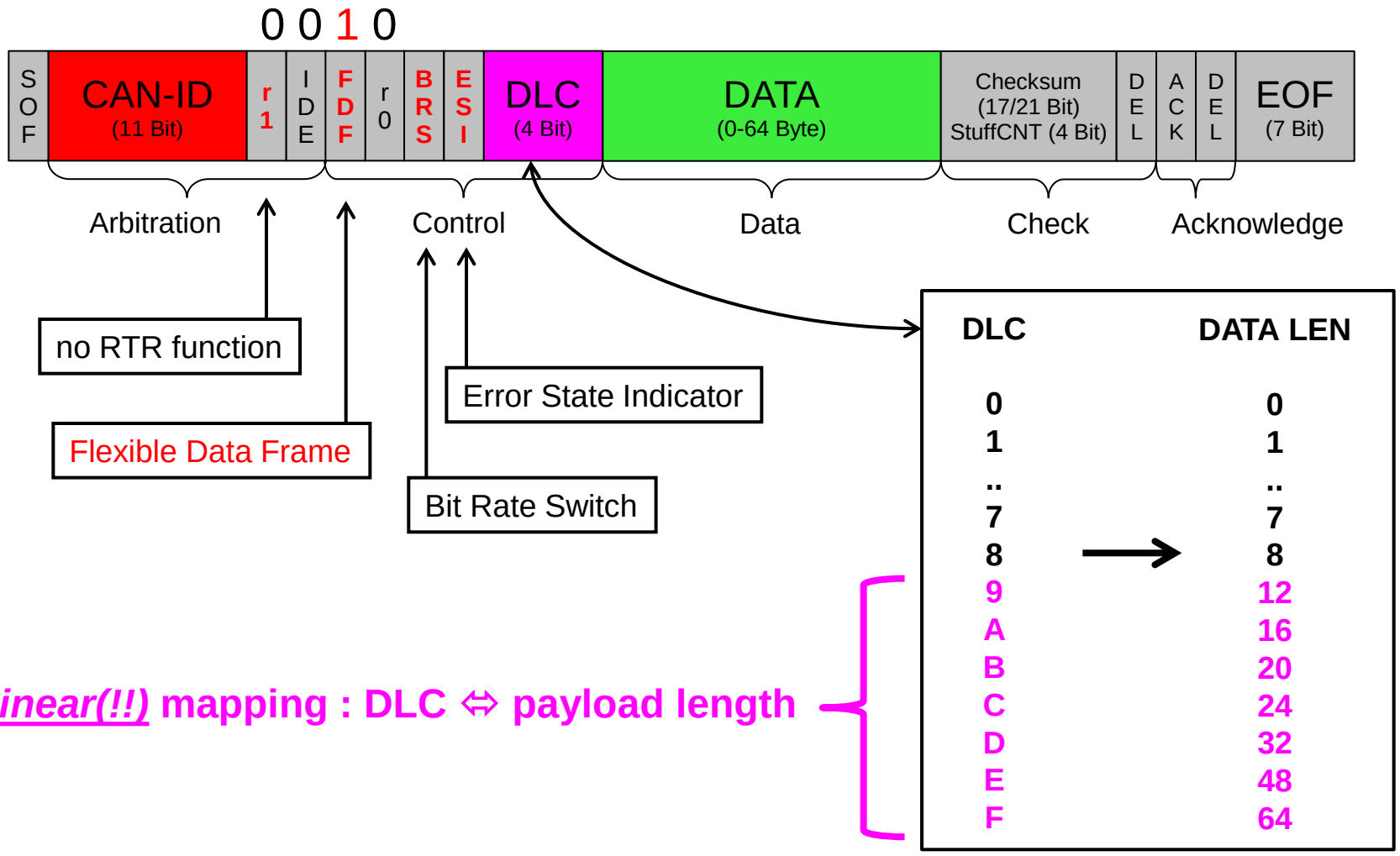
Dr. Oliver Hartkopp

Switching from CAN 2.0B to CAN FD by using the reserved bit



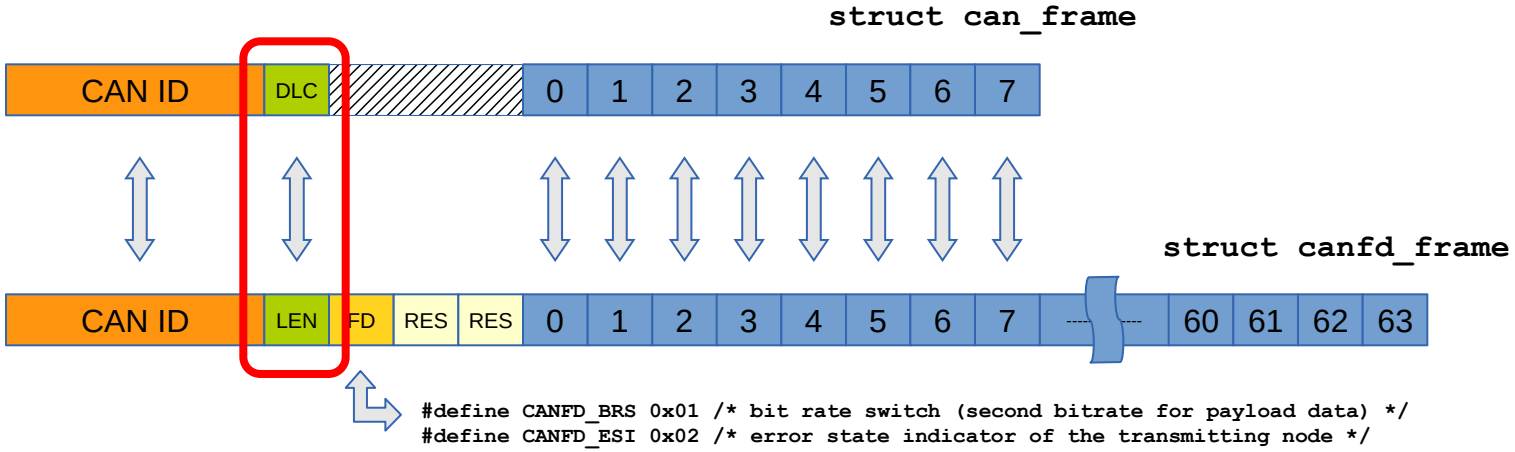


CAN FD – new bits and definitions in detail



Linux CAN FD length information and data structure

- DLC mostly has been used as plain payload length information (1:1 mapping)
- But CAN FD implements a non-linear length definition
- Introduce a structure element ‘len’ for CAN FD to preserve common usage
- The mapping of DLC ⇔ LEN and vice versa is done *invisible* in the CAN driver



Compatible data structure layout for CAN2.0B and CAN FD

- CAN2.0B data structure

```
struct can_frame {
        canid_t can_id;  /* 32 bit CAN_ID + EFF/RTR/ERR flags */
        __u8    can_dlc; /* frame payload length in byte (0 .. 8) */
        __u8    __pad;   /* padding */
        __u8    __res0;  /* reserved / padding */
        __u8    __res1;  /* reserved / padding */
        __u8    data[8] __attribute__((aligned(8)));
};
```

- CAN FD data structure

```
struct canfd_frame {
        canid_t can_id;  /* 32 bit CAN_ID + EFF/RTR/ERR flags */
        __u8    len;     /* frame payload length in byte (0 .. 64) */
        __u8    flags;   /* additional flags for CAN FD */
        __u8    __res0;  /* reserved / padding */
        __u8    __res1;  /* reserved / padding */
        __u8    data[64] __attribute__((aligned(8)));
};
```

Preserve common processing of length information

- Processing length information with CAN data structure

```
struct can_frame cframe;

for (i=0; i < cframe.can_dlc; i++)
    printf("%02X ", cframe.data[i]); /* print payload */
```

- Processing length information with CAN FD data structure

```
struct canfd_frame cframe;

for (i=0; i < cframe.len; i++)
    printf("%02X ", cframe.data[i]); /* print payload */

/* cframe.len = plain data length from 0 to 64 byte */
```

Preserve common processing of length information

- Processing length information with CAN data structure

```
struct can_frame cframe;

for (i=0; i < cframe.can_dlc; i++)
    printf("%02X ", cframe.data[i]); /* print payload */
```

- Processing length information with CAN FD data structure

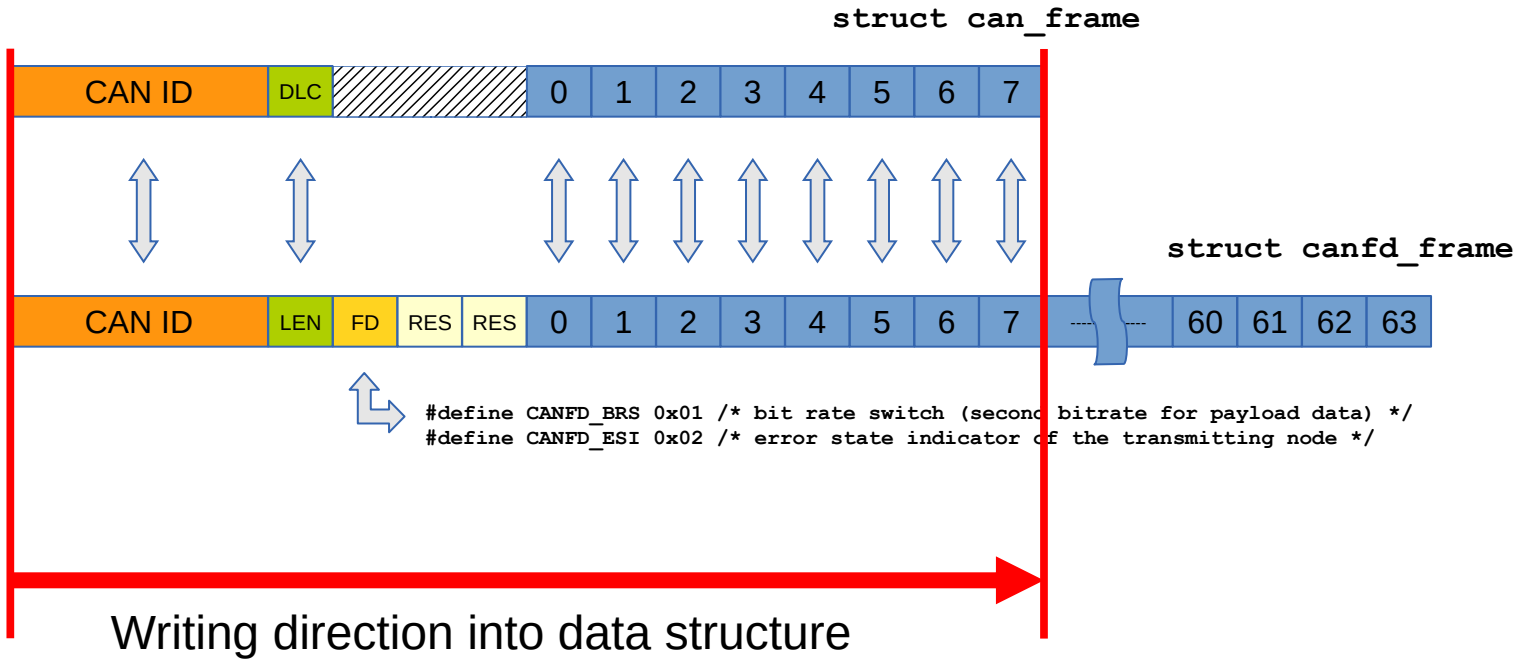
```
struct canfd_frame cframe;

for (i=0; i < cframe.len; i++)
    printf("%02X ", cframe.data[i]); /* print payload */

/* cframe.len = plain data length from 0 to 64 byte */
```

CAN FD data structure – dual use with Classic CAN layout

Writing CAN 2.0B data into a CAN FD data structure creates valid content.



How to activate CAN FD on a CAN_RAW socket

- Reading and writing CAN data structures

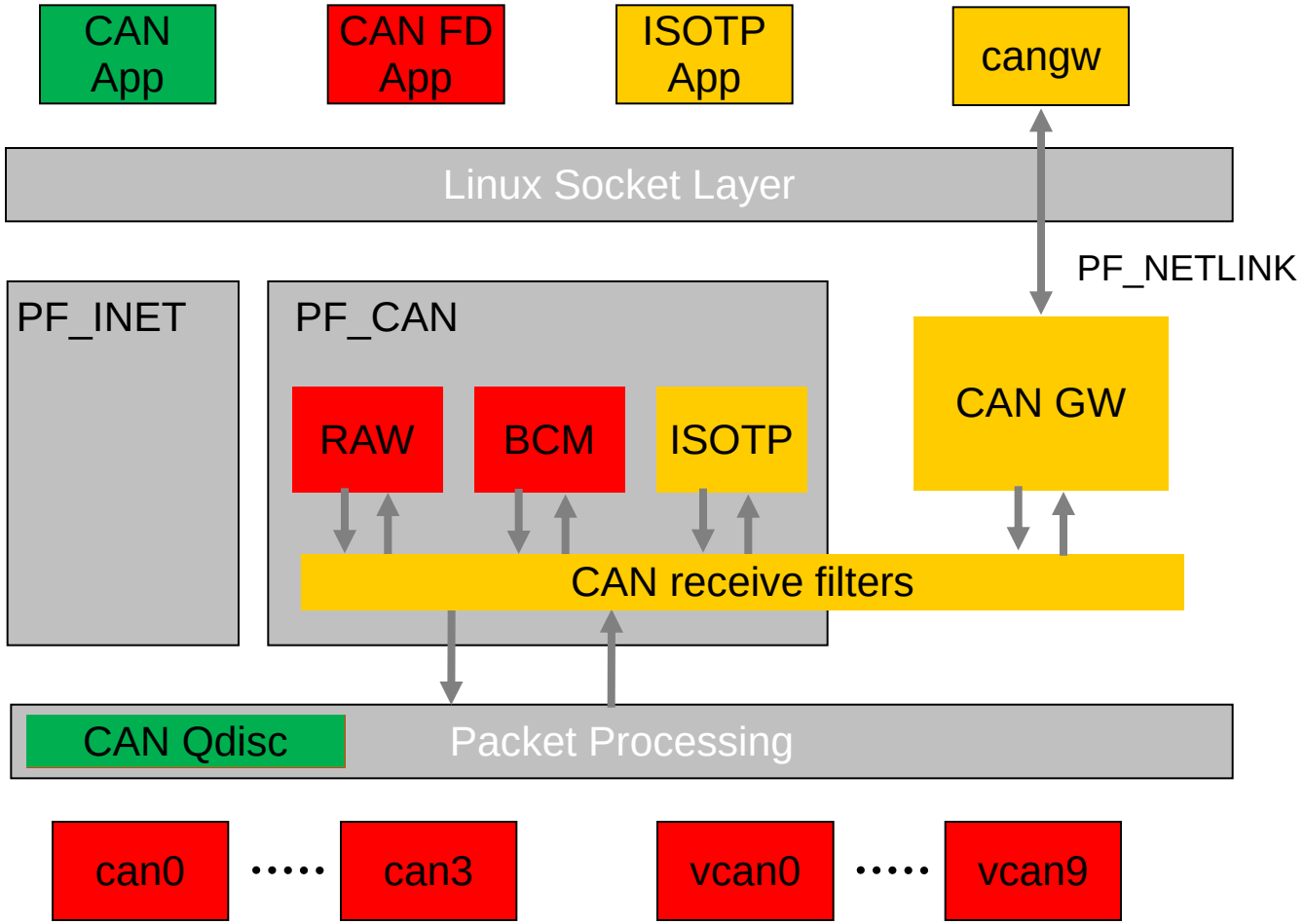
```
struct can_frame cframe;
int s = socket(PF_CAN, SOCK_DGRAM, CAN_RAW);
(..)
nbytes = read(s, &cframe, sizeof(struct can_frame));
```

- Switch the CAN_RAW socket into CAN FD mode with `setsockopt()` syscall

```
struct canfd_frame cframe;
int s = socket(PF_CAN, SOCK_DGRAM, CAN_RAW);
setsockopt(s, SOL_CAN_RAW, CAN_RAW_FD_FRAMES, ...);
(..)
nbytes = read(s, &cframe, sizeof(struct canfd_frame));
```

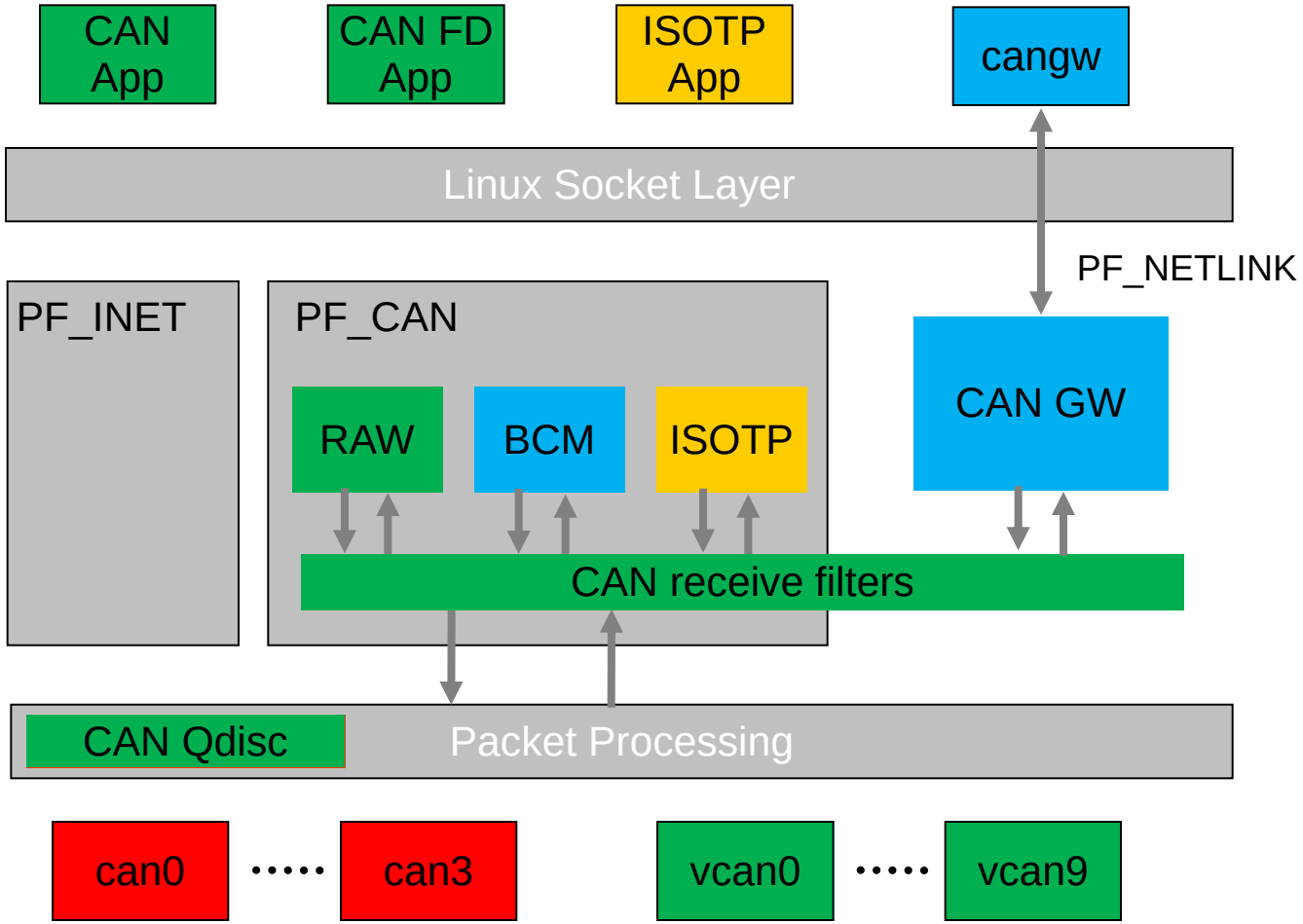
Impact of CAN FD to Linux and CAN applications

Depending on `struct can_frame`



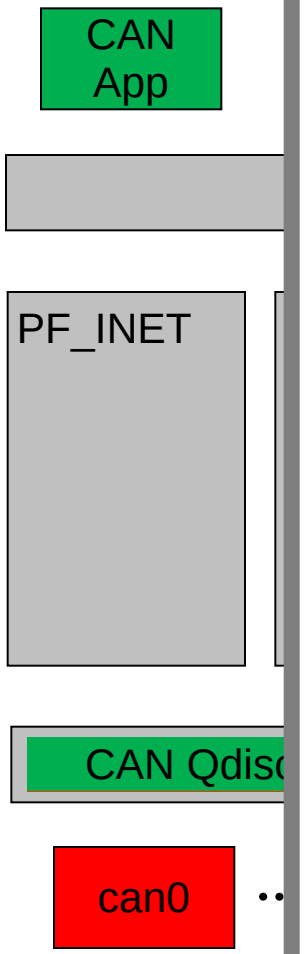
High impact
Indirect inpact
Supported

CAN FD support since Linux 3.6 (Summer 2012)



High impact
Indirect inpact
Supported
No use case?

CAN FD sup



CAN Newsletter Online

Home | Hardware | Software | Tools | Engineering

Home > Engineering > Standardization

Published 2012-07-03

Search

Editorial links

URLs
Linux organization
Volkswagen

News and reports
CAN FD specification
CAN FD protocol

Dossiers and features
25 years of CAN

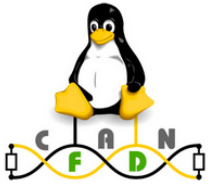
Knowledge
iCC proceedings
CAN FD protocol
CAN FD and CANopen

Additional information
CAN FD specification
CAN FD upstream patches
Linux CAN project
Linux CAN FD documentation

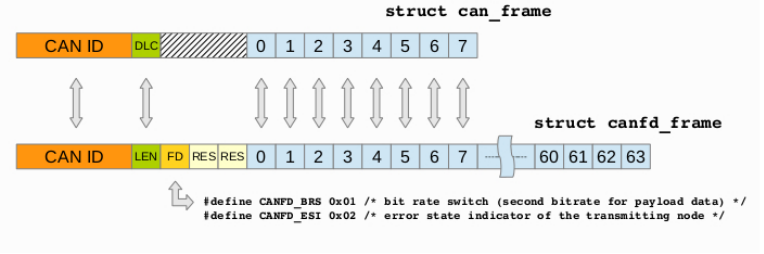
Save as PDF

Linux 3.6 supports CAN FD

The CAN FD (CAN with flexible data-rate) capable data structures and programming interfaces have been released for the Linux CAN sub-systems. This enables CAN application programmers to implement and run CAN FD applications on virtual CAN FD interfaces.



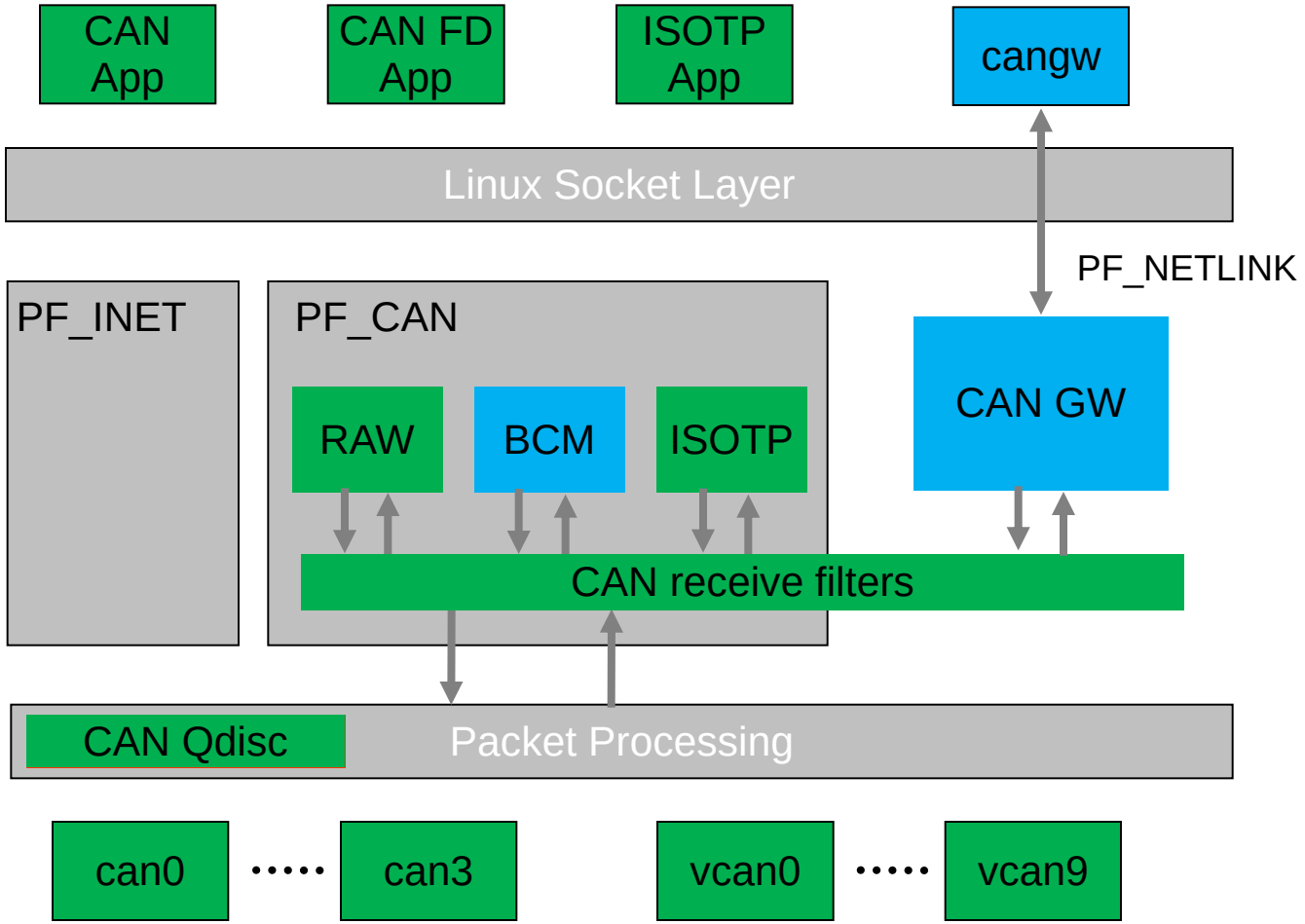
ON JUNE 19th 2012 THE LINUX NETWORK MAINTAINER David S. Miller pulled a set of six source code patches into the networking repository, which will be integrated in Linux version 3.6. The CAN FD patches from Oliver Hartkopp (Volkswagen, Germany) have been reviewed by the Linux CAN community and the sixth revision of these patches was finally approved. The integrated functionality to handle CAN FD frames defines the programming interfaces for application programmers as well as for CAN driver developers (when real CAN FD controllers become available). To preserve the binary compatibility for existing Linux CAN applications the socket programming interface has been extended by a CAN FD option, which is disabled by default. A CAN FD aware application may enable the CAN FD support on a per-socket basis, which allows sending and receiving CAN FD frames as well as "normal" CAN frames on this socket. The data structure for the CAN frame with its eight bytes of payload data was formerly assumed to be a fix point in CAN programming. With the introduction of CAN FD the payload data may consist of up to 64 bytes. In order to preserve the easy handling of CAN frames for application programmers a similar data structure for CAN FD frames has been defined:



The CAN FD data structure has a backward compatible layout, which allows processing all types of CAN frame. When a "normal" CAN frame content is read into the CAN FD structure, it can be accessed as a CAN FD frame. The CAN payload data length 'len' becomes a linear value from 0 to 64, which allows to preserve the known programming concepts, e.g. for loop programming statements. The mapping of the payload length to the DLC (data length code) field is supported by dedicated helper functions and is done on the CAN controller driver level only. This prevents the application programmer from cumbersome and error-prone mapping efforts. Currently, CAN FD applications and tools may be programmed an tested with the upgraded Virtual CAN (FD) interfaces only. When the real CAN FD controllers are released to the public, a second bit-rate configuration for CAN interfaces will be added to the Linux CAN driver infrastructure as well as the possibility to switch the then available CAN FD modes.

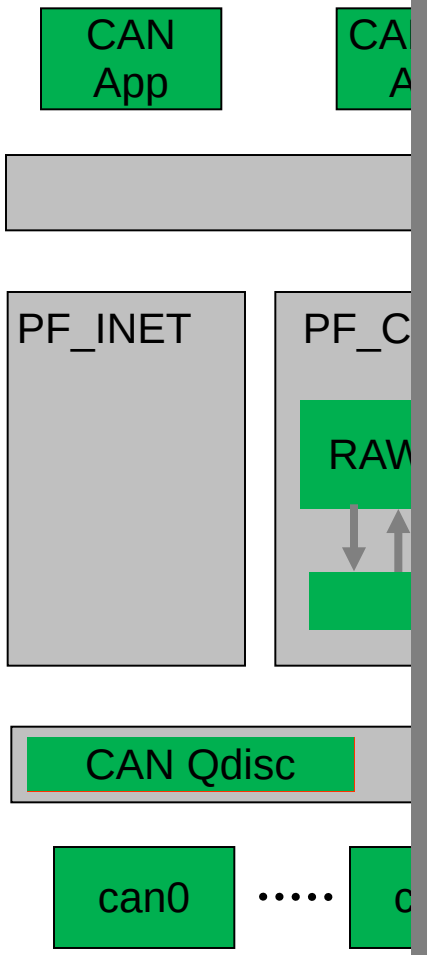
High impact
Indirect inpact
Supported
No use case?

Current CAN FD support since Linux 3.15 (Embedded W 2014)



High impact
Indirect inpact
Supported
No use case?

Current CAN FD … bedded W 2014)



CAN Newsletter Online

Home | Hardware | Software | Tools | Engineering

Home > Engineering > Standardization

Published 2014-05-13

Search

Editorial links

URLs
Peak
Linux

News and reports
CAN FD testing tool
CAN FD products
CAN FD at Embedded World
CAN FD interface boards
Processor module with Linux
Linux PLC
Linux 3.6 supports CAN FD

Knowledge
iCC proceedings
CAN FD protocol
CAN F and CANopen

CAN Newsletter (print)
CAN networking in Linux

Additional information
Linux CAN project
Can-utils (source code)
Can-utils (Debian package)
Linux CAN (FD) documentation

Embedded World

Save as PDF

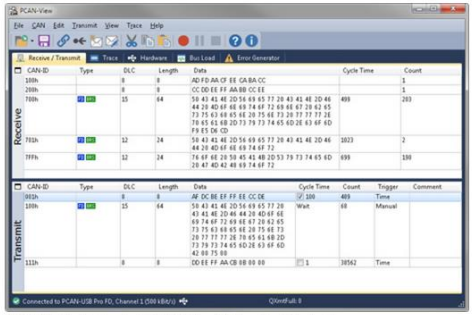
CAN FD Linux tools and driver infrastructure

At the opening day of the Embedded World 2014 a CAN FD plug fest connecting a Windows and a Linux system was presented. At the Peak-System booth, the different systems were connected using two PCAN-USB Pro FD adapters.

Configuration of a CAN FD controller with the 'ip' command from the latest 'iproute2' package (Linux screenshots: O. Hartkopp)

BASED ON THE CAN FD CAPABILITIES that were introduced in the Linux 3.6 CAN networking subsystem in Summer 2012, the CAN FD capable driver infrastructure will find its way into Linux 3.15. The donated PCAN-USB Pro FD adapter provides the full CAN FD functionalities, which enabled the Linux CAN community to discuss and implement the CAN FD extension for the unified Linux CAN driver infrastructure.

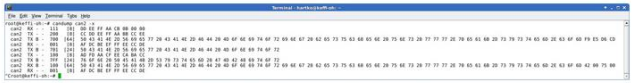
The CAN FD capability of the CAN controller is now exposed to the Linux system by the CAN driver, which enables additional CAN FD specific configuration options. These options allow to switch between classic CAN and CAN FD mode as well as the definition of the data bitrate (dbitrate) settings. The bitrate settings can be configured either by providing a single bitrate value (e.g. 1000000) or by a set of time quanta, segment value and jump width definitions.



CAN FD data displayed with PCAN-View (Photo: Peak-System)

At the booth the CAN FD adapters were spontaneously configured with 500 kbit/s for the arbitration bitrate and with 4000 kbit/s for the data bitrate, which led to an instant communication between the Windows and the Linux driven setup.

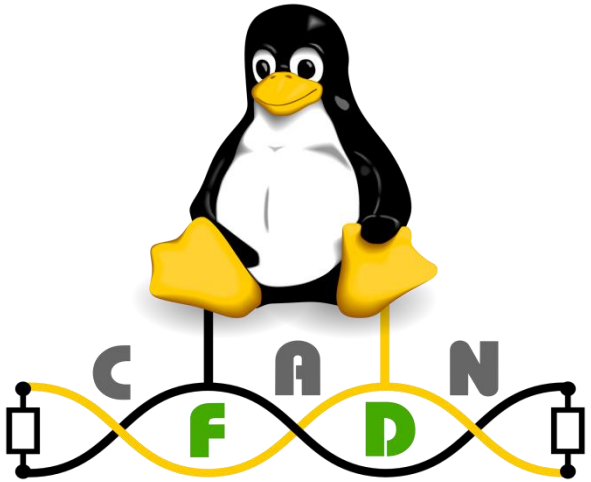
The existing Linux user-space tools to send, receive, store, and replay CAN traffic (aka 'can-utils') also confirmed their CAN FD capabilities in the interaction with the extended PCAN-View on Windows. The standard tool for Linux network configuration ('iproute2') will support the CAN FD specific options together with the release of the Linux 3.15 kernel and will automatically become part of common Linux distributions in the future.



CAN FD data displayed with 'can-dump' from the 'can-utils' package (Linux screenshots: O. Hartkopp)

With Linux 3.15, programming a CAN FD interface driver and using the CAN FD enabled network hardware becomes as easy as known from classic CAN interfaces. The 'can-utils' recently became an official Debian package for easy installation on Debian-based Linux distributions (Debian, Ubuntu, Linux Mint, etc.) with the existing software package managers. Alternatively the open source 'can-utils' can be downloaded from the Linux CAN community repository.

High impact
Indirect inpact
Supported
No use case?

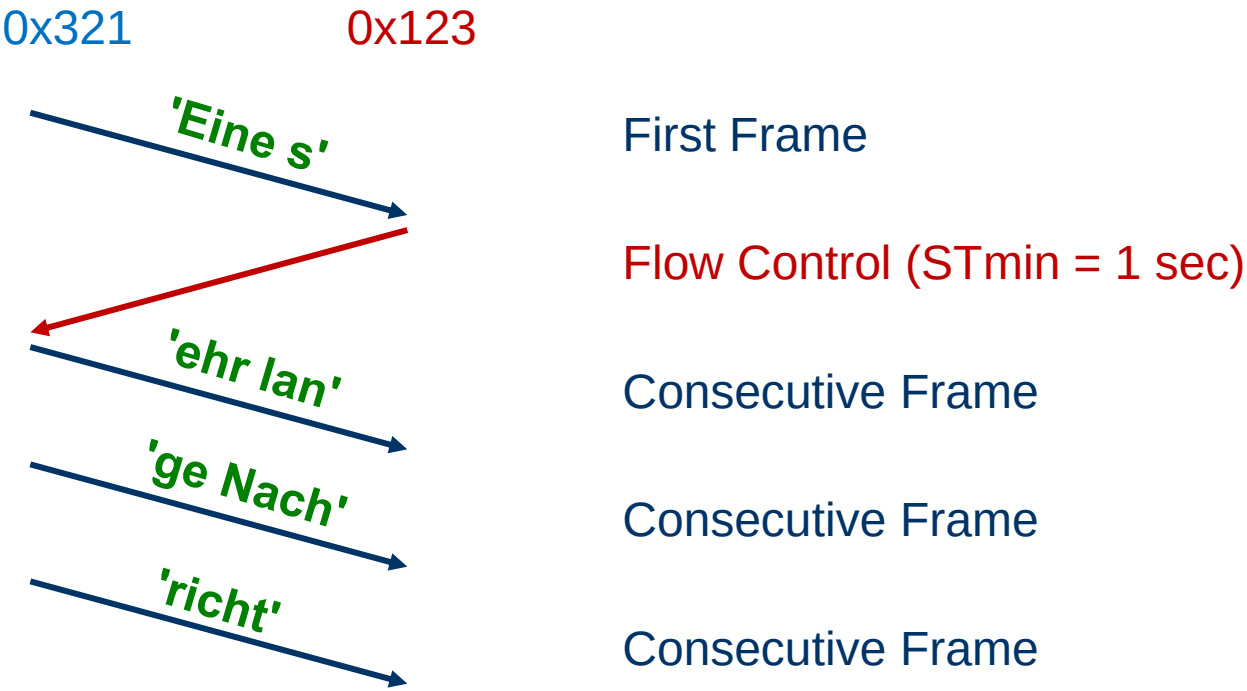


ISO 15765-2:2015 with CAN FD

Dr. Oliver Hartkopp

CAN transport protocol ISO 15765-2:2011

- Segmented transfer of application content
- Transfer up to 4095 bytes per ISO-TP PDU
- Bidirectional communication on two defined CAN IDs 0x321 0x123



Example source code

Creation of a point-to-point ISO 15765-2 transport channel

```
struct sockaddr_can addr;
char data[] = "Eine sehr lange Nachricht";        /* "a very long message" */


int s = socket(PF_CAN, SOCK_DGRAM, CAN_ISOTP);    /* create isotp socket instance */


addr.can_family = AF_CAN;                         /* address family AF_CAN */
addr.can_ifindex = if_nametoindex("can0")         /* CAN interface index for can0 */
addr.can_addr.tp.tx_id = 0x321;                   /* transmit on this CAN ID */
addr.can_addr.tp.rx_id = 0x123;                   /* receive on this CAN ID */


bind(s, (struct sockaddr *)&addr, sizeof(addr));  /* establish isotp communication */


write(s, data, strlen(data));                     /* sending of messages */
read(s, data, strlen(data));                      /* reception of messages */


close(s);                                         /* close socket instance */
```

ISO 15765-2:2015 with CAN FD support

- Segmented transports with First Frame (FF), Consecutive Frame (CF) and Flow Control (FC) are adapted to the first received CAN FD frame length

- For Single Frame (SF) PDUs handling based on CAN frame length
  - CAN frame length <= 8 byte: data length in first PCI byte, like CAN 2.0B
  - CAN frame length > 8 byte: set SF_DL=0 and data length in following byte

PCI Type	PCI layout	Byte 0	Byte 1	Byte 2	Byte 3	Byte 4	Byte 5	Byte 6
SF	0000LLLL	0 L	data	data	data	data	data	data
SF	00000000	0 0	L L	data	data	data	data	data

TX_DL = 8

```
vcan0  222  [02]  01 01
vcan0  222  [03]  02 01 02
vcan0  222  [04]  03 01 02 03
vcan0  222  [05]  04 01 02 03 04
vcan0  222  [06]  05 01 02 03 04 05
vcan0  222  [07]  06 01 02 03 04 05 06
vcan0  222  [08]  07 01 02 03 04 05 06 07
vcan0  222  [08]  10 08 01 02 03 04 05 06
vcan0  111  [03]  30 00 00
vcan0  222  [03]  21 07 08
```

SF PCI

TX_DL=8 → SF_DL data length in byte #1 (as 15765-2:2011)

One segmented message

TX_DL = 20

```
vcan0  222  [02]  01 01
vcan0  222  [03]  02 01 02
vcan0  222  [04]  03 01 02 03
vcan0  222  [05]  04 01 02 03 04
vcan0  222  [06]  05 01 02 03 04 05
vcan0  222  [07]  06 01 02 03 04 05 06
vcan0  222  [08]  07 01 02 03 04 05 06 07
vcan0  222  [12]  00 08 01 02 03 04 05 06 07 08 CC CC
vcan0  222  [12]  00 09 01 02 03 04 05 06 07 08 09 CC
vcan0  222  [12]  00 0A 01 02 03 04 05 06 07 08 09 0A
vcan0  222  [16]  00 0B 01 02 03 04 05 06 07 08 09 0A 0B CC CC CC
vcan0  222  [16]  00 0C 01 02 03 04 05 06 07 08 09 0A 0B 0C CC CC
vcan0  222  [16]  00 0D 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D CC
vcan0  222  [16]  00 0E 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E
vcan0  222  [20]  00 0F 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F CC CC CC
vcan0  222  [20]  00 10 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F 10 CC CC
vcan0  222  [20]  00 11 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F 10 11 CC
vcan0  222  [20]  00 12 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F 10 11 12
vcan0  222  [20]  10 13 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F 10 11 12
vcan0  111  [03]  30 00 00
vcan0  222  [02]  21 13
```

(mandatory padding)

SF PCI

SF_DL data length in byte #1 (as 15765-2:2011)

SF_DL escape value zero in byte #1

Data length shifted to byte #2

segmented message

ISO 15765-2:2015 with CAN FD support

- ‘Jumbo ISO-TP PDU Length’ definition in First Frame (FF)
  - PDU Length <= 4095 Byte: Data length in two PCI bytes (like CAN 2.0B)
  - PDU Length > 4095 Byte: set FF_DL=0 and data length in the following four bytes ⇨ allows a maximum PDU length of 2^{32}-1 (~4GB) bytes

PCI Type	PCI layout	Byte 0	Byte 1	Byte 2	Byte 3	Byte 4	Byte 5	Byte 6
FF	0001LLLL	1 L	L L	data	data	data	data	data
FF	00010000	1 0	0 0	L L	L L	L L	L L	data

ISO-TP CAN 2.0B / CAN FD comparison

CAN 2.0B and CAN FD throughput
with 2/4/8 Mbit/s data rate
STmin 500µs and 100µs
Blocksize 15, 30.000 bytes PDU

* = with bitrate setting (BRS) in CAN FD data section

```
bitrate 500000 sample-point 0.800
dbitrate 8000000 dsample-point 0.800

CAN2.0 08  (BS:15 # STmin:500 usec) : 30000 byte in 2.924670s => 10259 byte/s
CAN-FD 08  (BS:15 # STmin:500 usec) : 30000 byte in 2.930125s => 10238 byte/s
CAN-FD 08* (BS:15 # STmin:500 usec) : 30000 byte in 2.906859s => 10323 byte/s
CAN-FD 16  (BS:15 # STmin:500 usec) : 30000 byte in 1.378958s => 21770 byte/s
CAN-FD 16* (BS:15 # STmin:500 usec) : 30000 byte in 1.360648s => 22058 byte/s
CAN-FD 32  (BS:15 # STmin:500 usec) : 30000 byte in 0.790902s => 37974 byte/s
CAN-FD 32* (BS:15 # STmin:500 usec) : 30000 byte in 0.648899s => 46296 byte/s
CAN-FD 64  (BS:15 # STmin:500 usec) : 30000 byte in 0.625417s => 48000 byte/s
CAN-FD 64* (BS:15 # STmin:500 usec) : 30000 byte in 0.324803s => 92592 byte/s

CAN2.0 08  (BS:15 # STmin:100 usec) : 30000 byte in 1.461833s => 20533 byte/s
CAN-FD 08  (BS:15 # STmin:100 usec) : 30000 byte in 1.751986s => 17133 byte/s
CAN-FD 08* (BS:15 # STmin:100 usec) : 30000 byte in 1.149875s => 26109 byte/s
CAN-FD 16  (BS:15 # STmin:100 usec) : 30000 byte in 1.085210s => 27649 byte/s
CAN-FD 16* (BS:15 # STmin:100 usec) : 30000 byte in 0.544838s => 55147 byte/s
CAN-FD 32  (BS:15 # STmin:100 usec) : 30000 byte in 0.792370s => 37878 byte/s
CAN-FD 32* (BS:15 # STmin:100 usec) : 30000 byte in 0.265852s => 113207 byte/s
CAN-FD 64  (BS:15 # STmin:100 usec) : 30000 byte in 0.613622s => 48939 byte/s
CAN-FD 64* (BS:15 # STmin:100 usec) : 30000 byte in 0.130771s => 230769 byte/s
```

```
bitrate 500000 sample-point 0.800
dbitrate 4000000 dsample-point 0.800

CAN2.0 08  (BS:15 # STmin:500 usec) : 30000 byte in 2.921575s => 10270 byte/s
CAN-FD 08  (BS:15 # STmin:500 usec) : 30000 byte in 2.933282s => 10228 byte/s
CAN-FD 08* (BS:15 # STmin:500 usec) : 30000 byte in 2.911439s => 10305 byte/s
CAN-FD 16  (BS:15 # STmin:500 usec) : 30000 byte in 1.379933s => 21754 byte/s
CAN-FD 16* (BS:15 # STmin:500 usec) : 30000 byte in 1.364406s => 21994 byte/s
CAN-FD 32  (BS:15 # STmin:500 usec) : 30000 byte in 0.794492s => 37783 byte/s
CAN-FD 32* (BS:15 # STmin:500 usec) : 30000 byte in 0.661190s => 45385 byte/s
CAN-FD 64  (BS:15 # STmin:500 usec) : 30000 byte in 0.624796s => 48076 byte/s
CAN-FD 64* (BS:15 # STmin:500 usec) : 30000 byte in 0.324812s => 92592 byte/s

CAN2.0 08  (BS:15 # STmin:100 usec) : 30000 byte in 1.460029s => 20547 byte/s
CAN-FD 08  (BS:15 # STmin:100 usec) : 30000 byte in 1.748973s => 17162 byte/s
CAN-FD 08* (BS:15 # STmin:100 usec) : 30000 byte in 1.165946s => 25751 byte/s
CAN-FD 16  (BS:15 # STmin:100 usec) : 30000 byte in 1.085566s => 27649 byte/s
CAN-FD 16* (BS:15 # STmin:100 usec) : 30000 byte in 0.545156s => 55045 byte/s
CAN-FD 32  (BS:15 # STmin:100 usec) : 30000 byte in 0.792095s => 37878 byte/s
CAN-FD 32* (BS:15 # STmin:100 usec) : 30000 byte in 0.265145s => 113207 byte/s
CAN-FD 64  (BS:15 # STmin:100 usec) : 30000 byte in 0.614410s => 48859 byte/s
CAN-FD 64* (BS:15 # STmin:100 usec) : 30000 byte in 0.162909s => 185185 byte/s
```

```
bitrate 500000 sample-point 0.800
dbitrate 2000000 dsample-point 0.800

CAN2.0 08  (BS:15 # STmin:500 usec) : 30000 byte in 2.925522s => 10256 byte/s
CAN-FD 08  (BS:15 # STmin:500 usec) : 30000 byte in 2.933335s => 10228 byte/s
CAN-FD 08* (BS:15 # STmin:500 usec) : 30000 byte in 2.914893s => 10295 byte/s
CAN-FD 16  (BS:15 # STmin:500 usec) : 30000 byte in 1.379750s => 21754 byte/s
CAN-FD 16* (BS:15 # STmin:500 usec) : 30000 byte in 1.351478s => 22205 byte/s
CAN-FD 32  (BS:15 # STmin:500 usec) : 30000 byte in 0.791250s => 37926 byte/s
CAN-FD 32* (BS:15 # STmin:500 usec) : 30000 byte in 0.661982s => 45385 byte/s
CAN-FD 64  (BS:15 # STmin:500 usec) : 30000 byte in 0.625395s => 48000 byte/s
CAN-FD 64* (BS:15 # STmin:500 usec) : 30000 byte in 0.328604s => 91463 byte/s

CAN2.0 08  (BS:15 # STmin:100 usec) : 30000 byte in 1.459521s => 20562 byte/s
CAN-FD 08  (BS:15 # STmin:100 usec) : 30000 byte in 1.749743s => 17152 byte/s
CAN-FD 08* (BS:15 # STmin:100 usec) : 30000 byte in 1.172328s => 25597 byte/s
CAN-FD 16  (BS:15 # STmin:100 usec) : 30000 byte in 1.085260s => 27649 byte/s
CAN-FD 16* (BS:15 # STmin:100 usec) : 30000 byte in 0.548291s => 54744 byte/s
CAN-FD 32  (BS:15 # STmin:100 usec) : 30000 byte in 0.792734s => 37878 byte/s
CAN-FD 32* (BS:15 # STmin:100 usec) : 30000 byte in 0.329611s => 91185 byte/s
CAN-FD 64  (BS:15 # STmin:100 usec) : 30000 byte in 0.614448s => 48859 byte/s
CAN-FD 64* (BS:15 # STmin:100 usec) : 30000 byte in 0.225412s => 133333 byte/s
```

ISO-TP CAN 2.0B / CAN FD comparison results

ISO-TP payload data rate with 30.000 bytes ‚Jumbo PDUs‘
STmin 100µs, Blocksize 15

• CAN 2.0B (8 Byte Frames, 500 kBit/s)	20.547 byte/s
• CAN FD (8 Byte Frames, 500 kBit/s no BRS)	17.162 byte/s
• CAN FD (64 Byte Frames, 500 kBit/s no BRS)	48.859 byte/s
• CAN FD (64 Byte Frames, 500 kBit/s / 2MBit/s)	133.333 byte/s
• CAN FD (64 Byte Frames, 500 kBit/s / 4MBit/s)	185.185 byte/s
• CAN FD (64 Byte Frames, 500 kBit/s / 8MBit/s)	230.769 byte/s

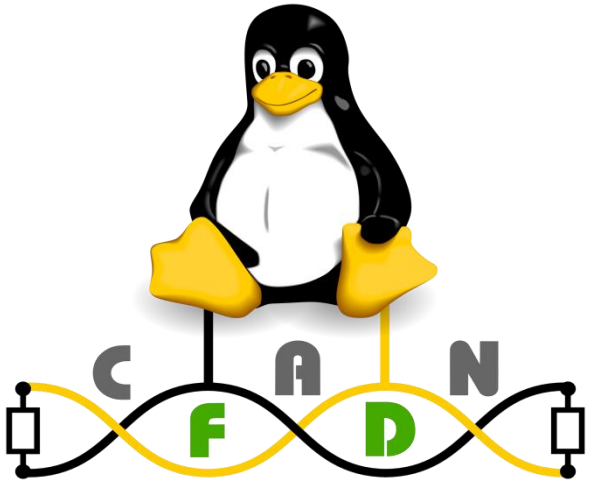
ISO-TP for CAN FD hands-on

New ISO-TP link layer socket options for CAN FD specific configuration

```
struct can_isotp_ll_options {
        __u8 mtu;       /* CAN / CAN FD frames switch */
        __u8 tx_dl;     /* 8, 12, 16, 20, 24, 32, 48, 64 */
        __u8 tx_flags; /* BRS switch */
};
```

Download and build ISO-TP implementation (tools already support CAN FD)

- `git clone https://github.com/hartkopp/can-isotp-modules.git`
- `cd can-isotp-modules/net/can/`
- `./make_isotp.sh`
- `sudo insmod ./can-isotp.ko`



CAN FD Integration in Linux Distributions and Hardware

Dr. Oliver Hartkopp

Out-Of-The-Box CAN Support

- Debian-based distributions with allmodconfig Linux Kernel
- Invoke `apt-get install can-utils` for CAN FD capable tools

(removed Logos of different Debian-based Linux distributions)

CAN hardware drivers

- CAN Controller IP Cores: FlexCAN, MSCAN, XILINX, C_CAN, D_CAN, M_CAN, AT91, Blackfin, CC770, SJA1000, MCP251x, etc.
- PC Hardware: ISA, PCI, PCMCIA, SPI, USB
- Embedded ‘Maker’ Hardware

(removed Logos / Photos of different CAN related manufacturers/products)

Generic CAN hardware configuration

- ‘ip’ tool from iproute2 package (standard network configuration tool)

- CAN bitrate configuration by
  (1) Bitrate, Sampling Point, Synchronization Jump Width
  (2) ISO: time quanta (tq), propagation segment (prop_seg), phase buffer segments (phase_seg1 phase_seg2)

- Controller specific bitrate calculation inside Linux kernel
- Switches for Listen-Only, Triple-Sampling, One-Shot-Mode, etc

- Second bitrate setting for CAN FD data bitrate
- Switches for CAN FD and CAN FD non-ISO Mode

Examples

can0:
USB2CAN
(8devices.com)

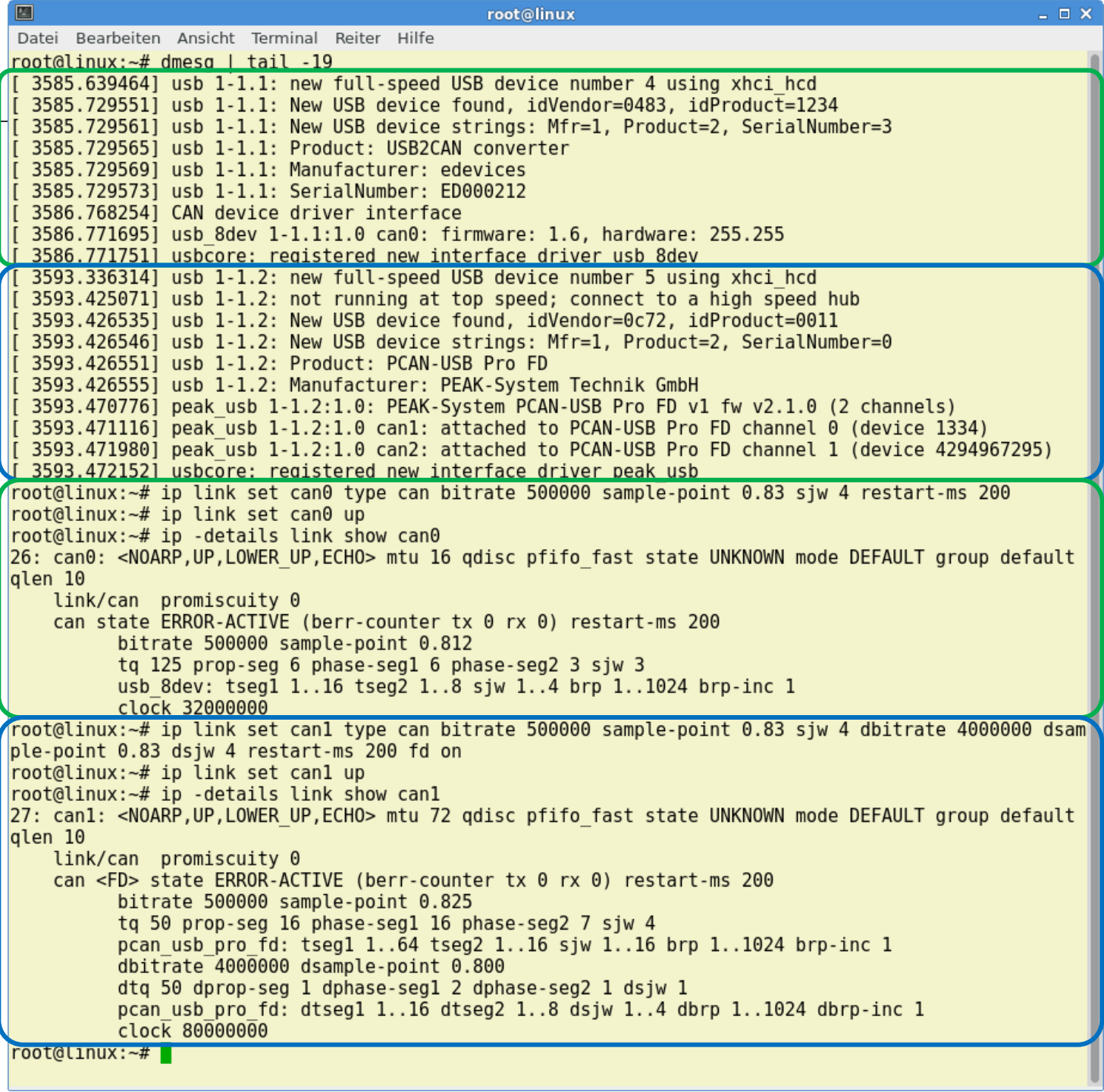
500kbit/s SP 83%

can1/can2:
PCAN USB pro FD
(PEAK System)

500kbit/s SP 83%

4MBit/s SP 83%

CAN FD enabled



Example configuration USB2CAN (8devices)

```
root@linux
Datei  Bearbeiten  Ansicht  Terminal  Reiter  Hilfe
root@linux:~# dmesg | tail -19
[ 3585.639464] usb 1-1.1: new full-speed USB device number 4 using xhci_hcd
[ 3585.729551] usb 1-1.1: New USB device found, idVendor=0483, idProduct=1234
[ 3585.729561] usb 1-1.1: New USB device strings: Mfr=1, Product=2, SerialNumber=3
[ 3585.729565] usb 1-1.1: Product: USB2CAN converter
[ 3585.729569] usb 1-1.1: Manufacturer: edevices
[ 3585.729573] usb 1-1.1: SerialNumber: ED000212
[ 3586.768254] CAN device driver interface
[ 3586.771695] usb_8dev 1-1.1:1.0 can0: firmware: 1.6, hardware: 255.255
[ 3586.771751] usbcore: registered new interface driver usb_8dev
```

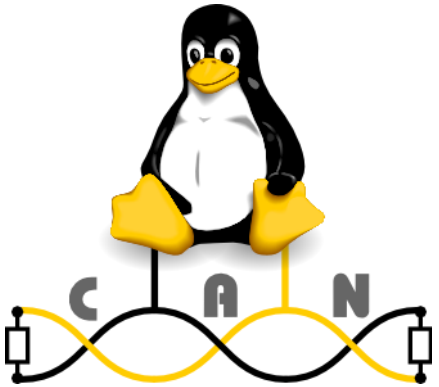
```
root@linux:~# ip link set can0 type can bitrate 500000 sample-point 0.83 sjw 4 restart-ms 200
root@linux:~# ip link set can0 up
root@linux:~# ip -details link show can0
26: can0: <NOARP,UP,LOWER_UP,ECHO> mtu 16 qdisc pfifo_fast state UNKNOWN mode DEFAULT group default
qlen 10
    link/can  promiscuity 0
    can state ERROR-ACTIVE (berr-counter tx 0 rx 0) restart-ms 200
          bitrate 500000 sample-point 0.812
          tq 125 prop-seg 6 phase-seg1 6 phase-seg2 3 sjw 3
          usb_8dev: tseg1 1..16 tseg2 1..8 sjw 1..4 brp 1..1024 brp-inc 1
          clock 32000000
```

Example configuration PCAN USB pro FD (PEAK system)

```
[ 3593.336314] usb 1-1.2: new full-speed USB device number 5 using xhci_hcd
[ 3593.425071] usb 1-1.2: not running at top speed; connect to a high speed hub
[ 3593.426535] usb 1-1.2: New USB device found, idVendor=0c72, idProduct=0011
[ 3593.426546] usb 1-1.2: New USB device strings: Mfr=1, Product=2, SerialNumber=0
[ 3593.426551] usb 1-1.2: Product: PCAN-USB Pro FD
[ 3593.426555] usb 1-1.2: Manufacturer: PEAK-System Technik GmbH
[ 3593.470776] peak_usb 1-1.2:1.0: PEAK-System PCAN-USB Pro FD v1 fw v2.1.0 (2 channels)
[ 3593.471116] peak_usb 1-1.2:1.0 can1: attached to PCAN-USB Pro FD channel 0 (device 1334)
[ 3593.471980] peak_usb 1-1.2:1.0 can2: attached to PCAN-USB Pro FD channel 1 (device 4294967295)
[ 3593.472152] usbcore: registered new interface driver peak_usb
```

```
root@linux:~# ip link set can1 type can bitrate 500000 sample-point 0.83 sjw 4 dbitrate 4000000 dsam
ple-point 0.83 dsjw 4 restart-ms 200 fd on
root@linux:~# ip link set can1 up
root@linux:~# ip -details link show can1
27: can1: <NOARP,UP,LOWER_UP,ECHO> mtu 72 qdisc pfifo_fast state UNKNOWN mode DEFAULT group default
qlen 10
    link/can  promiscuity 0
    can <FD> state ERROR-ACTIVE (berr-counter tx 0 rx 0) restart-ms 200
          bitrate 500000 sample-point 0.825
          tq 50 prop-seg 16 phase-seg1 16 phase-seg2 7 sjw 4
          pcan_usb_pro_fd: tseg1 1..64 tseg2 1..16 sjw 1..16 brp 1..1024 brp-inc 1
          dbitrate 4000000 dsample-point 0.800
          dtq 50 dprop-seg 1 dphase-seg1 2 dphase-seg2 1 dsjw 1
          pcan_usb_pro_fd: dtseg1 1..16 dtseg2 1..8 dsjw 1..4 dbrp 1..1024 dbrp-inc 1
          clock 80000000
root@linux:~#
```

Questions?




```
$> cat linux/MAINTAINERS | grep -B 2 -A 14 Hartkopp

CAN NETWORK LAYER
M:      Oliver Hartkopp <socketcan@hartkopp.net>
M:      Marc Kleine-Budde <mkl@pengutronix.de>
L:      linux-can@vger.kernel.org
W:      https://github.com/linux-can
T:      git git://git.kernel.org/pub/scm/linux/kernel/gut/mkl/linux-can.git
T:      git git://git.kernel.org/pub/scm/linux/kernel/gut/mkl/linux-can-next.git
S:      Maintained
F:      Documentation/networking/can.txt
F:      net/can/
F:      include/linux/can/core.h
F:      include/uapi/linux/can.h
F:      include/uapi/linux/can/bcm.h
F:      include/uapi/linux/can/raw.h
F:      include/uapi/linux/can/gw.h

$>
```