VOLKSWAGEN

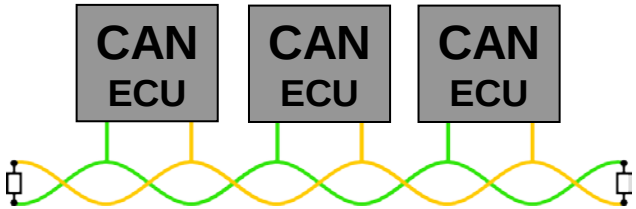
AKTIENGESELLSCHAFT

CAN & CAN FD inside AVTP Frames (IEEE 1722 Automotive)

Dr. Oliver Hartkopp

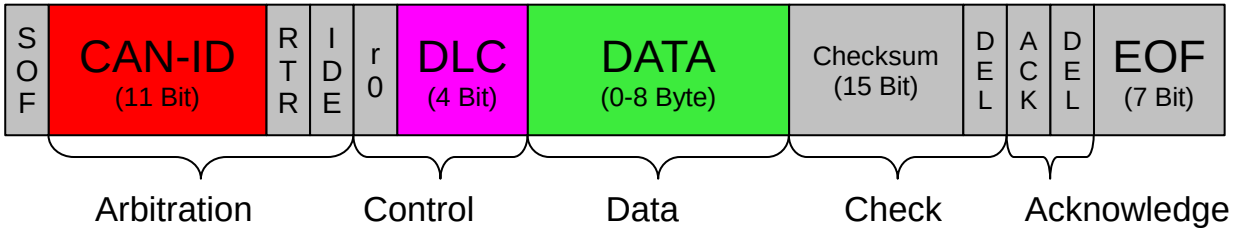
The known Controller Area Network Protokoll

- Media access via CSMA/CR



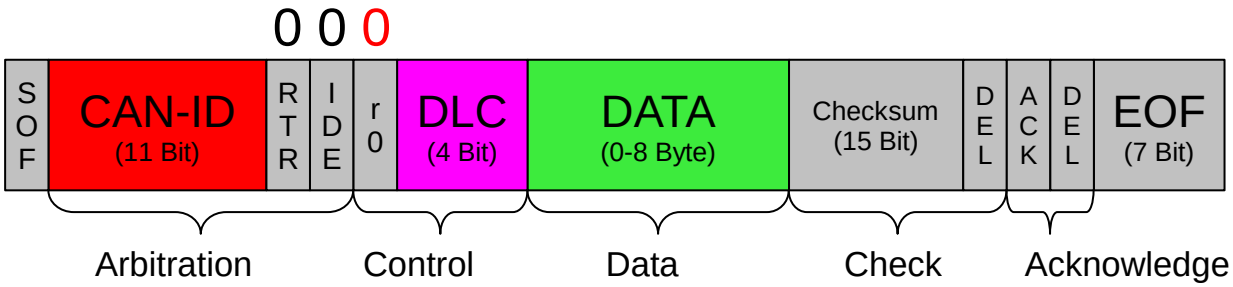
- Contentaddressing (CAN Identifier, CAN Bus)

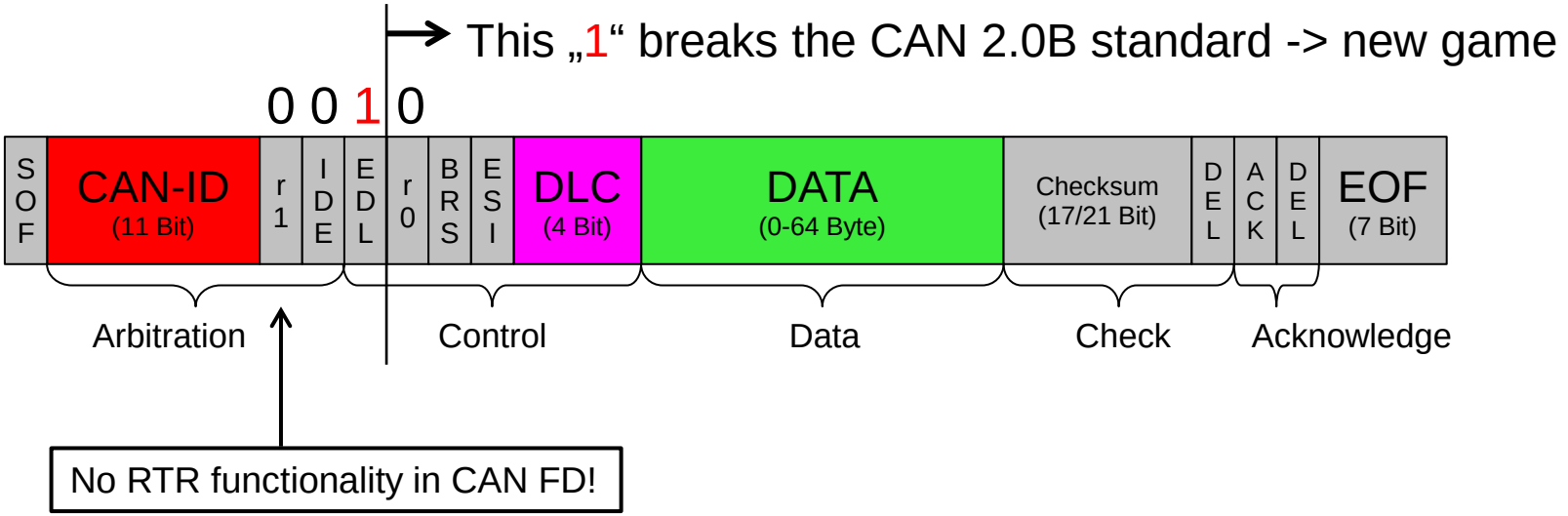
- Structure of a CAN Message – here “Basic CAN” with 11 bit identifier:



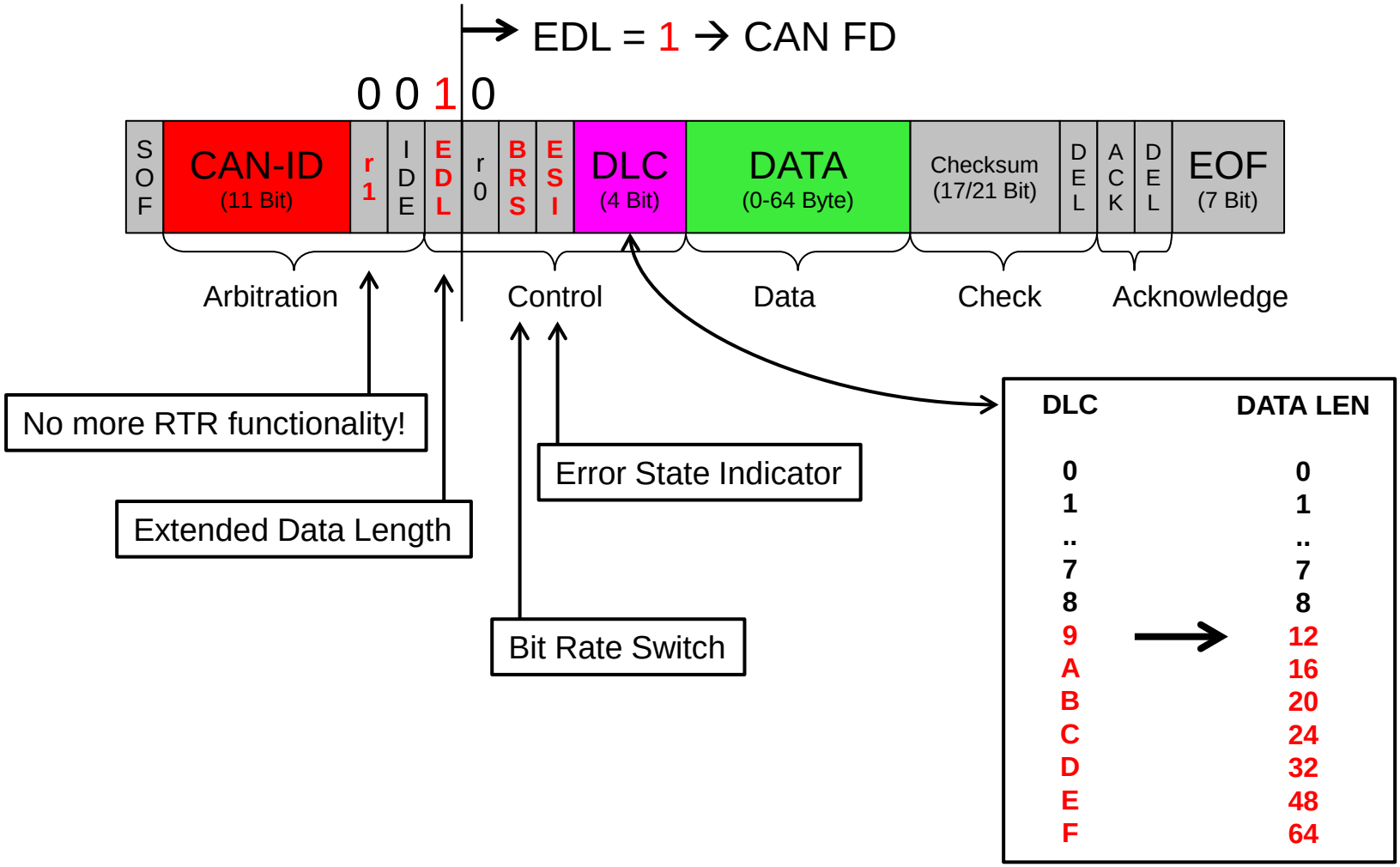
- Simplified: [CAN Identifier] [Data length] [Data 0..8]

CAN FD – Switching to the new protocol

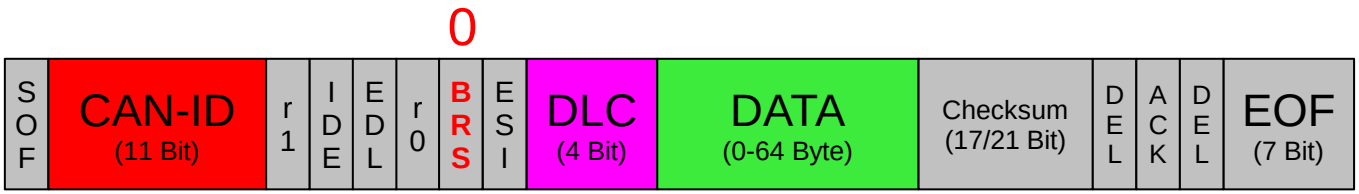


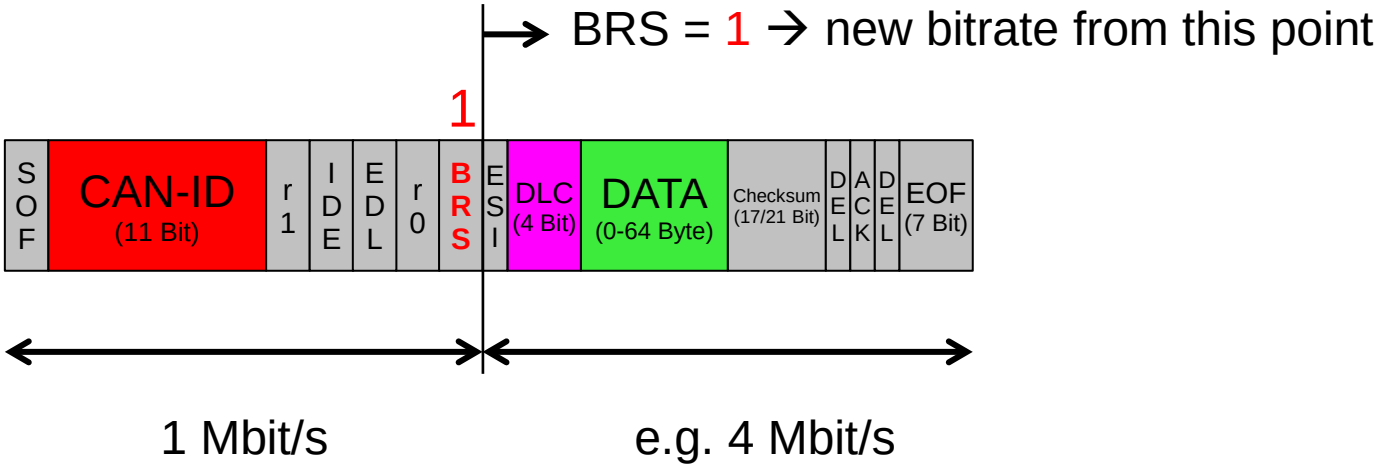


CAN FD – new bits and their meaning



CAN FD – second data rate for the payload data





Arbitration & data transfer performed with different bitrates and samplingpoints

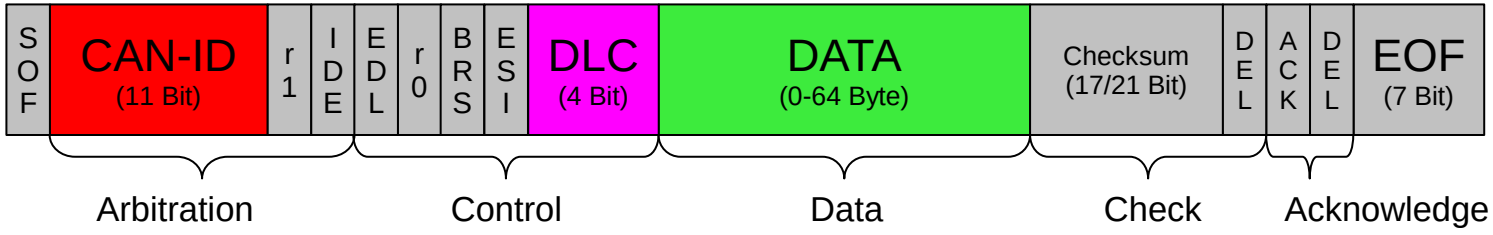
Possible CAN frame representations

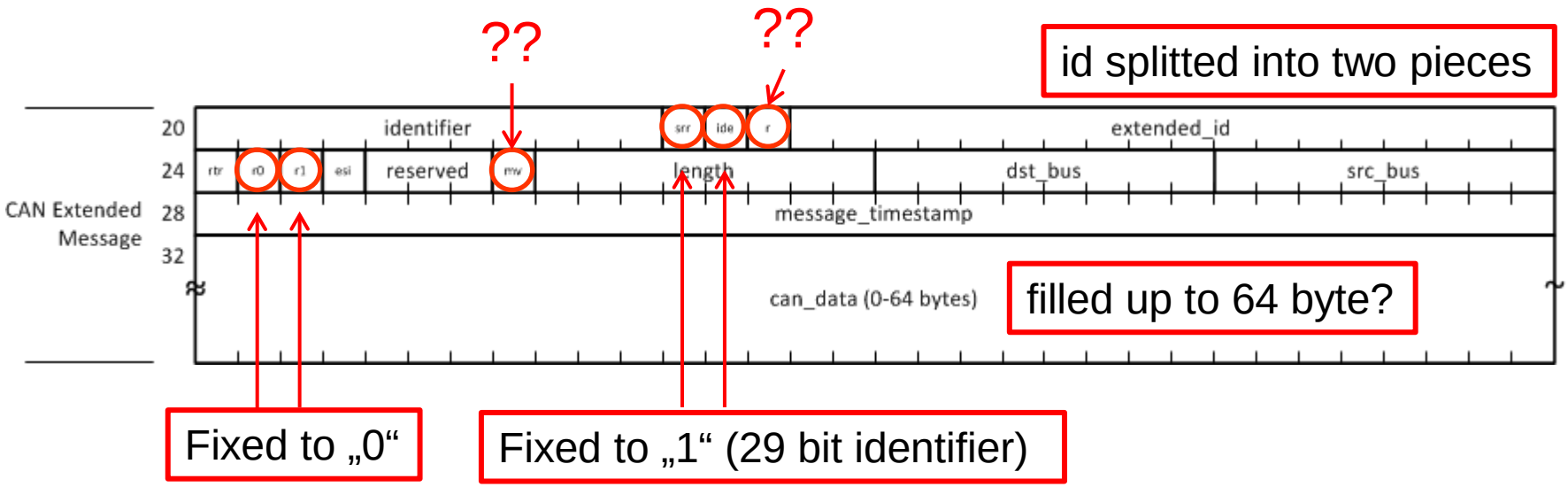
extended identifier	RTR	EDL	BRS	data length	Frame type
0	0	0	0	0 - 8	Basic CAN Frame (11 bit ID)
1	0	0	0	0 – 8	Extended CAN Frame (29 bit ID)
0	1	0	0	0	Basic CAN RTR Frame (11 bit ID)
1	1	0	0	0	Extended CAN RTR Frame (29 bit ID)
0	0	1	0	0 – 64	CAN FD Frame (11 bit ID)
1	0	1	0	0 – 64	CAN FD Frame (29 bit ID)
0	0	1	1	0 – 64	CAN FD Frame (11 bit ID, fast)
1	0	1	1	0 – 64	CAN FD Frame (29 bit ID, fast)

No warranty – but i did my very best …

Problems with the current CAN frame proposals

The physical representation on the wire is put into the ethernet frame …





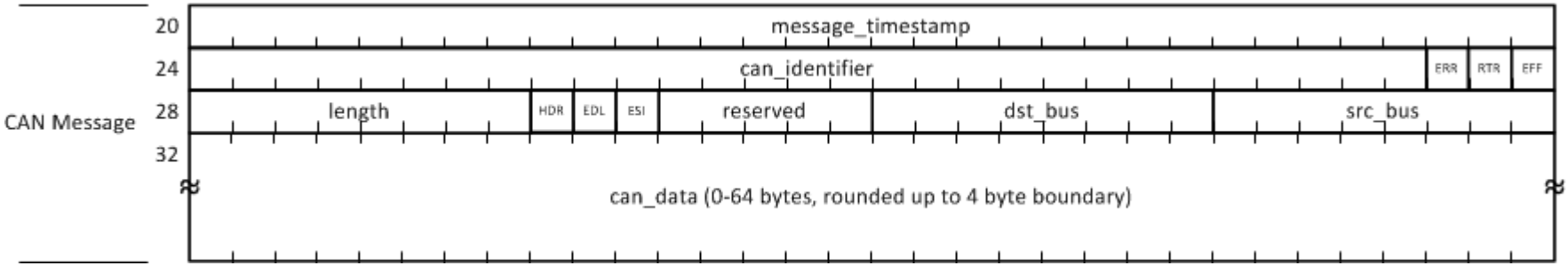
Problems when implementing the current proposal

- Bits that are not available in the system are added artificially to the etherframe
  - Bit values are only seen on the physical layer and not in CAN controller regs
  - 29 bit identifiers are split up in two pieces (pointless & costly bit shiftings)
  - No benefits from adding physical layer data bits – a working CAN controller bitstream engine must be available in the system anyway (CRC checks, etc).

- Two definitions that are not able to reproduce all CAN frame representations
  - „CAN Base“ has an 11 bit identifier (and fixed 8 bytes available for payload)
  - „CAN Extended“ has an 29 bit identifier AND up to 64 bytes of payload, which is a CAN FD feature(!)
  - CAN FD (EDL, BRS, ESI) has nothing to do with the identifier length (11/29)

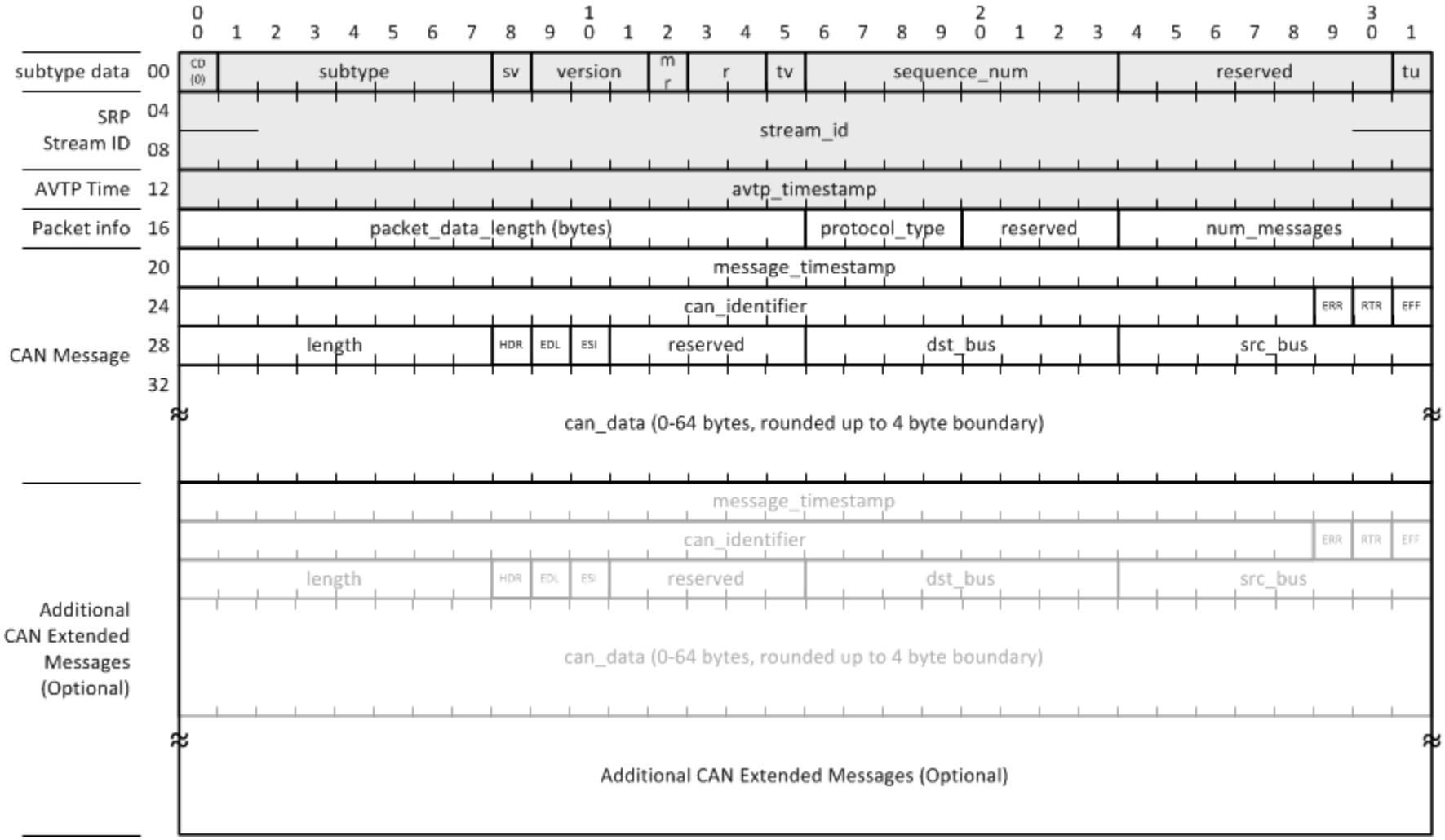
Proposal for a common data structure for CAN frames

- Define one data structure that copes with 11/29 bit IDs, RTR, CAN FD, …
- Omit pointless physical layer information (e.g. use data length instead of DLC)
- Define the payload data length block (can_data) upon the needed space
- Round up the payload data block length to 4 (0 →0, 3→4, 4→4, 5→8, 64→64)



- RTR: Remote Transmission Request
- EFF: Extended Frame Format (29 bit ID)
- ERR: CAN controller error messages (optional)
- length: plain payload data length (0 .. 64 [bytes])
- can_identifier: 11/29 bit ID depending on EFF
- EDL: Extended Data Length (CAN FD)
- HDR: High Data Rate (CAN FD)
- ESI: Error State Indicator (CAN FD)
- can_data: payload data block aligned to 4 byte
- message_timestamp: CAN rx timestamp in ns

Entire CAN data frame encapsulation

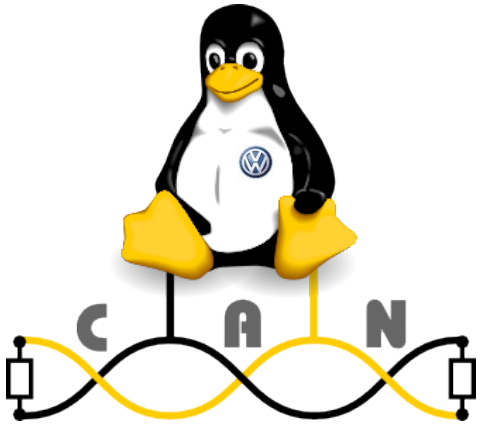


Reference to Linux CAN FD data structure

Similar requirements in Linux CAN networking lead to this community approved data structure with 32 bit space for the CAN-ID, real data length information and additional flags for CAN FD. The **__res0** and **__res1** bytes can be used for **src_bus** and **dst_bus** which additionally allows an elegant and performant Linux implementation for IEEE 1722a (simple memcopy operations, no bitwise ops).

```
/**
 * struct canfd_frame - CAN flexible data rate frame structure
 * @can_id: CAN ID of the frame and CAN_*_FLAG flags, see canid_t definition
 * @len:    frame payload length in byte (0 .. CANFD_MAX_DLEN)
 * @flags:  additional flags for CAN FD
 * @__res0: reserved / padding
 * @__res1: reserved / padding
 * @data:   CAN FD frame payload (up to CANFD_MAX_DLEN byte)
 */
struct canfd_frame {
            canid_t can_id;  /* 32 bit CAN_ID + EFF/RTR/ERR flags */
            __u8    len;     /* frame payload length in byte */
            __u8    flags;   /* additional flags for CAN FD */
            __u8    __res0;  /* reserved / padding */
            __u8    __res1;  /* reserved / padding */
            __u8    data[CANFD_MAX_DLEN] __attribute__((aligned(8)));
};
```

See: http://git.kernel.org/?p=linux/kernel/git/davem/net-next.git;a=blob_plain;f=include/linux/can.h;hb=HEAD

Many thanks!

```
$> cat linux-3.2/MAINTAINERS | grep -B 2 -A 11 Hartkopp

CAN NETWORK LAYER
M:      Oliver Hartkopp <socketcan@hartkopp.net>
L:      linux-can@vger.kernel.org
W:      http://gitorious.org/linux-can
T:      git git://gitorious.org/linux-can/linux-can-next.git
S:      Maintained
F:      net/can/
F:      include/linux/can.h
F:      include/linux/can/core.h
F:      include/linux/can/bcm.h
F:      include/linux/can/raw.h
F:      include/linux/can/gw.h

$> _
```