University of Stuttgart
Institute of Parallel and Distributed Systems

Distributed Systems

Fachpraktikum / Lab-Course

Software-Defined and Time-Sensitive Networking

Tutorial: CAN-Bus

Frank Dürr

Summer Term 2023

IPVS

Agenda

- Overview
- Medium Access Control (MAC) Protocol
- CAN Frames Format
- SocketCAN

CAN: Controller Area Network

Overview

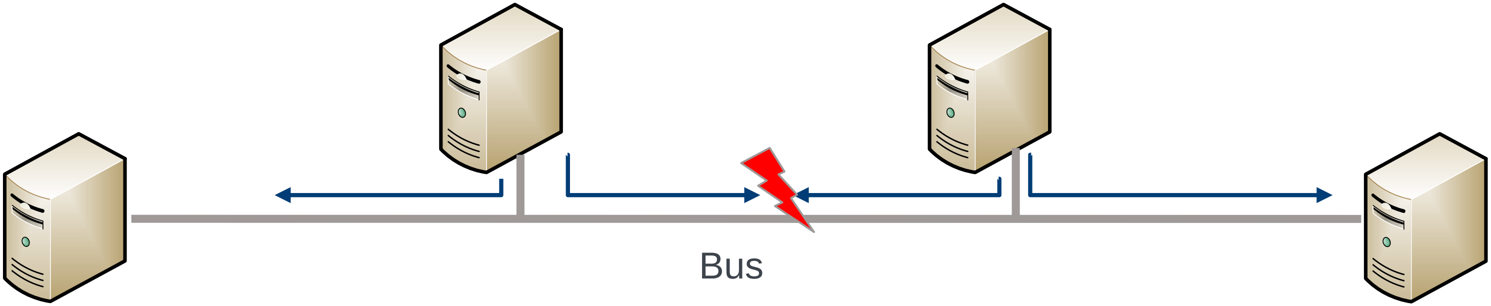
- Fieldbus technology
- One major application field: automotive
- Typical data rates:
  - 1 Mbps for up to 40 m distance
  - 125 kbps for up to 150 m distance
- Bus topology
  - Many stations connected to one physical medium (cable)

Medium Access Control (MAC)

Problem: Collisions

Problem: collisions if more than one station sends at a time

- Medium Access Control (MAC) controls access to the shared physical medium
  - CSMA (Carrier Sense Multiple Access): “listen before talk”
  - Ethernet (IEEE 802.3): CSMA/CD (Collision Detection)
  - CAN: CSMA/CR (Collision Resolution)



Medium Access Control (MAC)

CSMA

If a station wants to send: Carrier sensing:

- Listen on medium
  - if medium free: send frame immediately
  - if medium not free:
    - **1-persistent CSMA**: continue sensing and start immediately when medium is sensed free
      - 1-persistent CSMA is used by Ethernet and **CAN**
    - Non-persistent CSMA: sense again after random time

Medium Access Control (MAC)

CSMA/CD

CSMA + Collision Detection used by Ethernet: while sending, detect collisions

- If no collision is detected until end of frame: successful transmission
- If collision is detected:
  - Wait random back-off time from an interval
    - Multiple collisions exponentially increase the size of the back-off interval, i.e. decrease the chance that multiple stations choose same back-off time
  - Start again by carrier sensing
    - Possibly again collision if multiple stations chose same back-off time

→ Time to success unbounded (possibly total frame loss since stations give up after a certain number of collisions)

→ CSMA/CD not suited for hard/firm real-time communication

Medium Access Control (MAC)

CSMA/CR (1)

CAN uses CSMA/CR (Collision Resolution)

- If a collision occurs, it will be resolved in favor of the station with highest priority

➔ Deterministically bounded latency only for highest-priority frames

➔ Unbounded latency for lower-priority frames

Deterministic bounds can be ensured for multiple streams by combination with a TDMA scheme (Time-Triggered CAN / TTCAN)

- Time slots assigned to stations
- CSMA/CR within each time slot

Not discussed further here

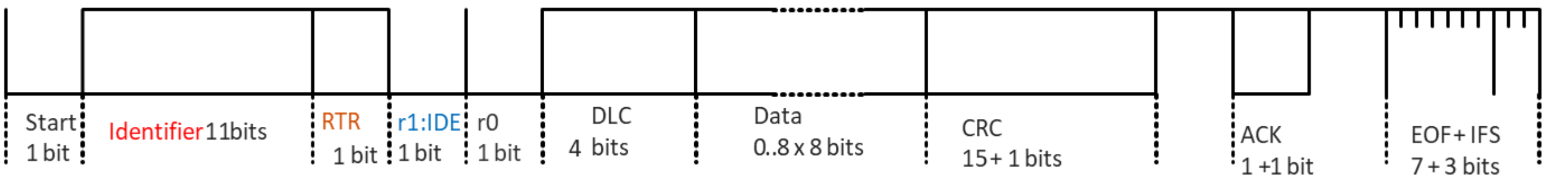
Medium Access Control (MAC)

CSMA/CR (2)

- After carrier sensing for free medium, concurrent senders are synchronized
  - Will start transmitting their CAN frames simultaneously
- Dominant bits overwrite recessive bits transmitted simultaneously
  - 0 bits are dominant
  - 1 bits are recessive
- While sending, sender checks (reads) bits on bus
- Sender gives up as soon as one of its recessive bits has been overwritten by a dominant bit of other sender (other sender has higher priority)
- Tries again after carrier sensing senses free medium again

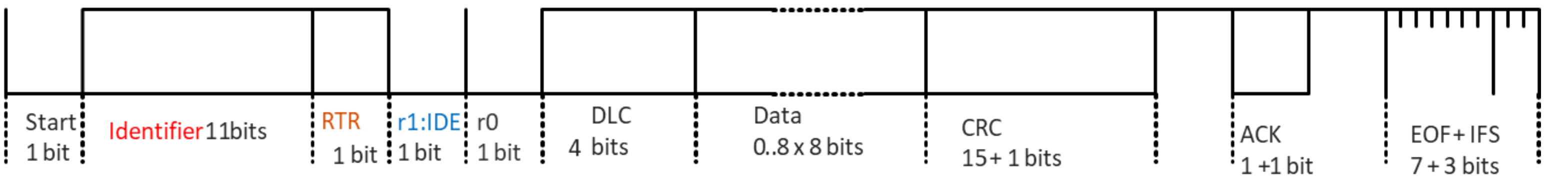
➔ Sender with more leading dominant bits transmits successfully

CAN Frame (1)



- Identifier identifies data sent or requested
  - Note: no sender or receiver station (MAC) addresses as in Ethernet!
  - 11 bit (standard format) or 29 bit (extended format) identifiers
  - The smaller the identifier, the higher the priority (0 bits are dominant!)
  - Example:
    - ID of sensor value “motor speed”: 00000000001
    - ID of sensor value “temperature”: 00000000010
    - Speed value data and data requests have highest priority
    - Speed value data has higher priority than speed value data requests

CAN Frame (2)



- Remote Transmission Request (RTR) set to 1 to request data, 0 to send data
  - Data source with data corresponding to identifier will reply
- Data Length Code (DLC): length of data
- Data: 0 to 8 Byte
- CRC: checksum to detect bit errors
- ACK
  - Sender sends 1, receiver overwrites with 0 to acknowledge correct data
  - If sender reads 1: failure; else correct transmission

Latency Bounds of MAC

For highest priority, latency until bus can be successfully accessed is bounded:

$$d_{max} = \frac{max.\,frame\ length}{bus\ bit\ rate}$$

At 1 Mbit/s and max. frame lengths of

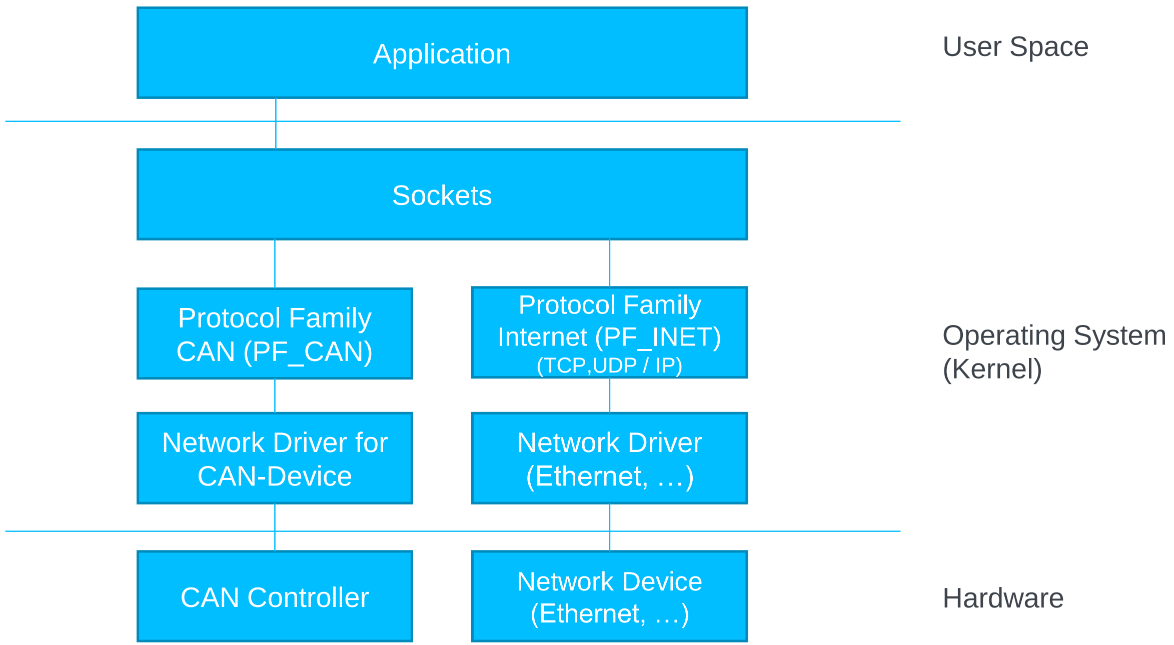
- 130 bit: d_{max}= 130 µs
- 154 bit: d_{max}= 154 µs

(simplified since CAN uses bit-stuffing: after 5 same bits, there must be a complementary bit inserted)

SocketCAN

What is the operating system API for applications to send CAN messages?

- In POSIX, the answer is clear: sockets!



SocketCAN

Overview

- Open-source network stack for CAN
- Contributed by Volkswagen Research to the Linux kernel
- Multiple processes can access the CAN interface
- Using the common socket API
  - Protocol family: PF_CAN
- Raw sockets
  - Must hand over a complete CAN frame incl. header & data to kernel
- Support for ISO-TP (Transport Layer)
  - Send messages > 8 bytes over CAN
  - We will send single frames over raw sockets

SocketCAN

Data Types (1)

```
/*
 * Controller Area Network Identifier structure
 * - bit 0-28   : CAN identifier (11/29 bit)
 * - bit 29     : error message frame flag (0 = data frame,
 *                1 = error message)
 * - bit 30     : remote transmission request flag
 *                (1 = rtr frame; 0 otherwise)
 * - bit 31     : frame format flag (0 = standard 11 bit,
 *                1 = extended 29 bit)
 */
 canid_t canid;
```

SocketCAN

Data Types (2)

```
/*
 * CAN frame with the following data fields:
 * - canid_t can_id  : the can id
 * - uint8_t can_dlc : data length
 * - uint8_t data[8] : data
 */
struct can_frame {
        canid_t can_id;   /* 32 bit CAN_ID + EFF/RTR/ERR flags */
        __u8    can_dlc;  /* data length code: 0 .. 8 */
        __u8    data[8] __attribute__((aligned(8)));
};
```

SocketCAN

Sending and Receiving CAN Frames

```c
int cansock;
struct can_frame frame;


ssize_t nsent = write(cansock, &frame, sizeof(frame));


ssize_t len = read(csock, &frame, sizeof(frame));
```

Note: `frame` is a complete CAN frame

- Includes CAN ID
- Don't need to specify addresses with `sendto()`, `recvfrom()`

SocketCAN

Creating a CAN Socket

```
int cansock;

cansock = socket(PF_CAN, SOCK_RAW, CAN_RAW));
```

SocketCAN

Binding a CAN Socket (1)

```
struct ifreq ifr;
struct sockaddr_can addr;

// Determine interface index of given CAN interface with name ifname.
strncpy(ifr.ifr_name, ifname, IFNAMSIZ-1);
ifr.ifr_name[IFNAMSIZ - 1] = '\0';
if (ioctl(cansock, SIOCGIFINDEX, &ifr) < 0)
    return ERROR;
```

SocketCAN

Binding a CAN Socket (2)

```c
// Bind to interface.
addr.can_family = AF_CAN;
addr.can_ifindex = ifr.ifr_ifindex;
if (bind(cansock, (struct sockaddr *) &addr, sizeof(addr)) < 0)
    return ERROR;
else
    return SUCCESS;
```

Questions?