VOLKSWAGEN

AKTIENGESELLSCHAFT

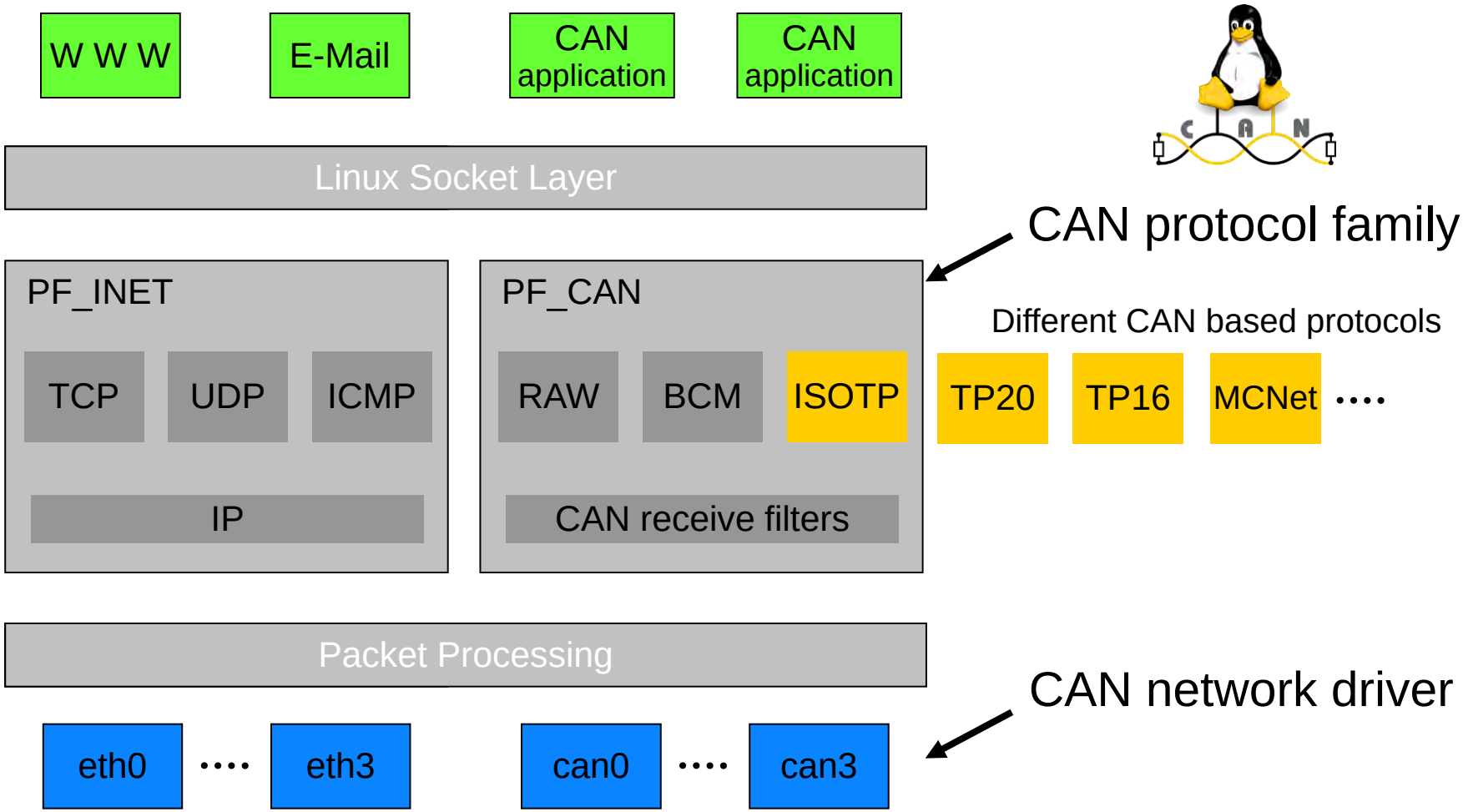
The CAN Subsystem of the Linux Kernel

CAN in Automation - 13th international CAN Conference

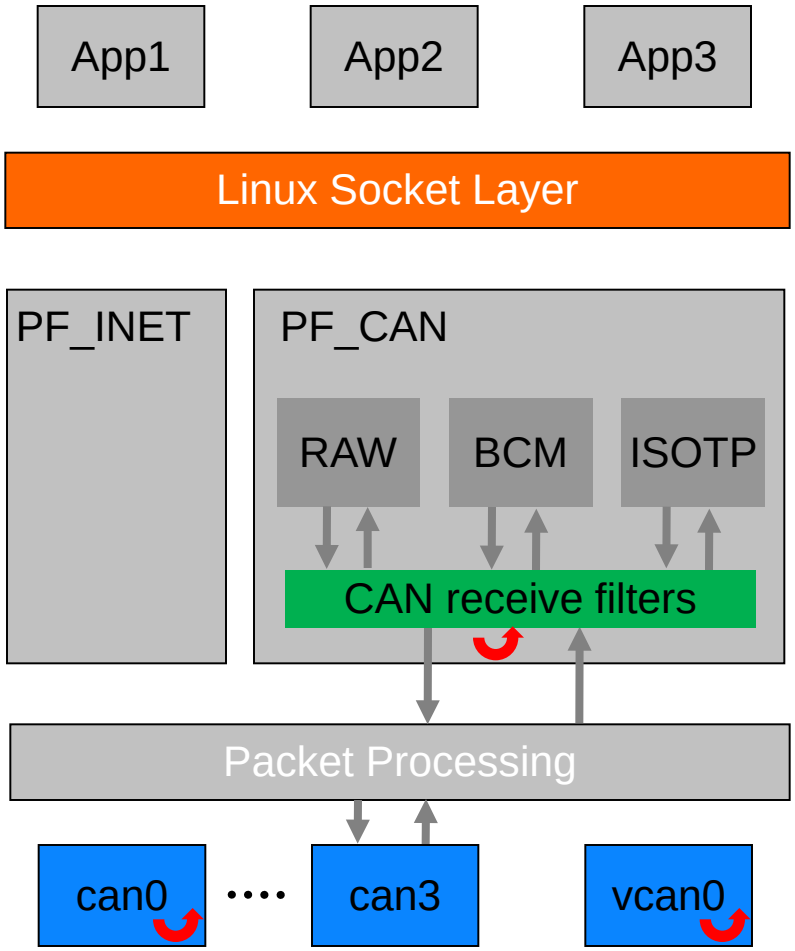
Dr. Oliver Hartkopp

ELEKTRONIK & FAHRZEUG

Protocol family for the Controller Area Network (PF_CAN)



Highlights of the protocol family PF_CAN



Standard BSD network socket programming interface

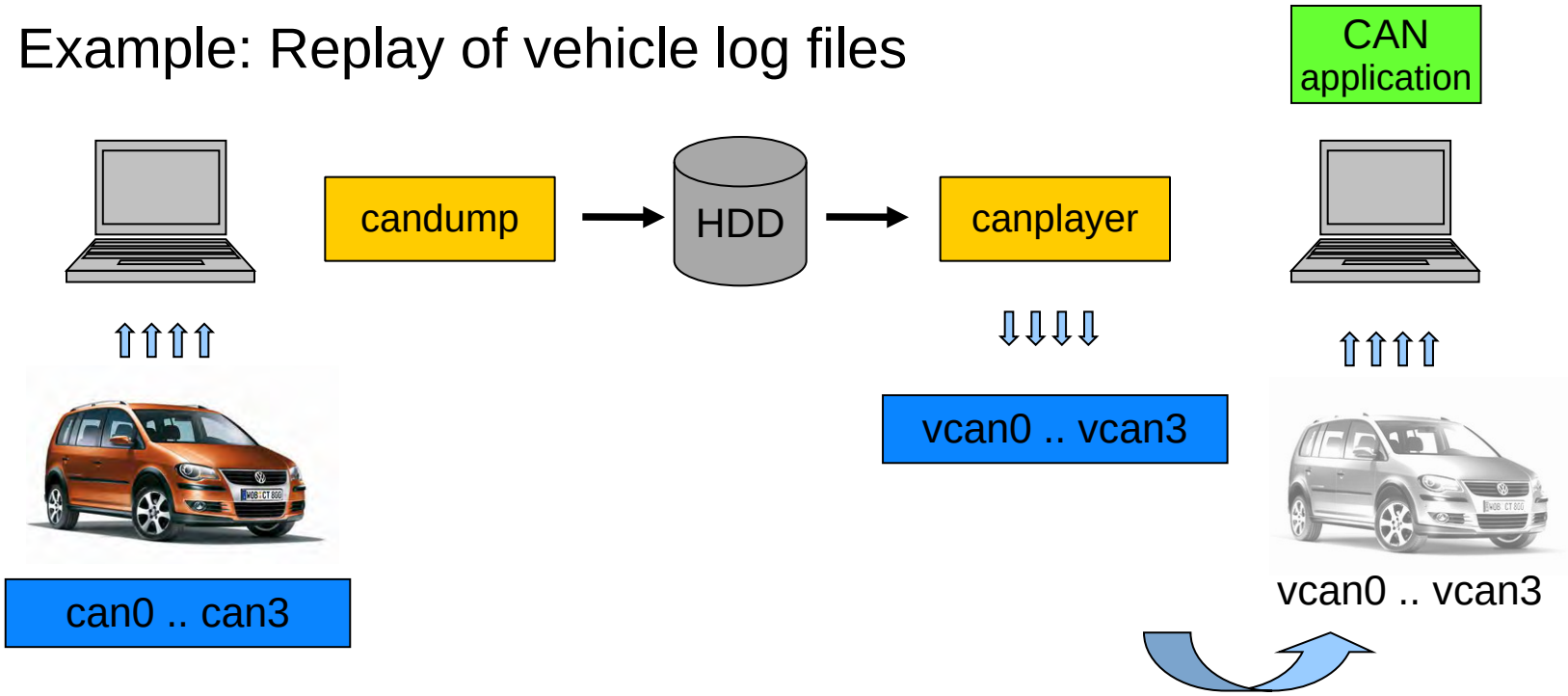
Receive filter lists handled inside a software interrupt (Linux NET_RX softirq)

network device driver model

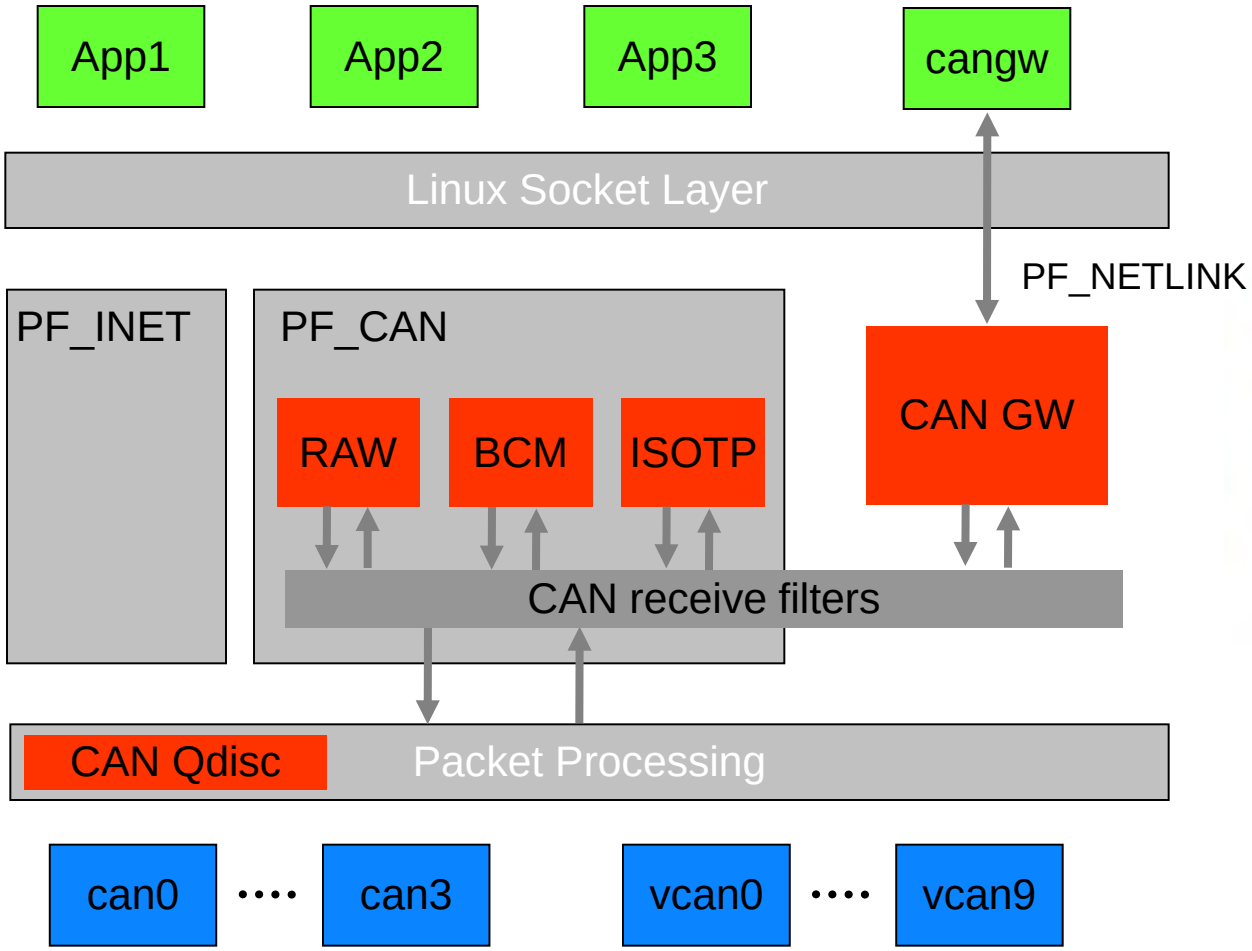
Network transparency:
local echo of **sent** CAN frames on successful transmission

Virtual CAN network device driver (vcan)

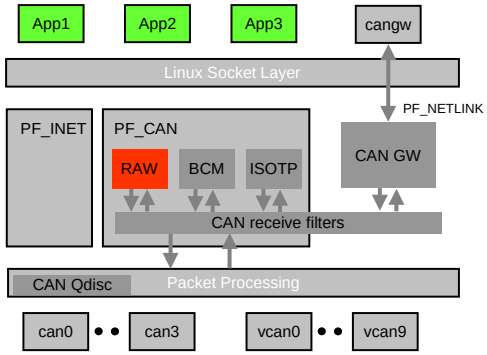
- No need for real CAN hardware (available since Linux 2.6.25)
- Local echo of sent CAN frames ‘loopback device’
- vcan instances can be created at run-time
- Example: Replay of vehicle log files



CAN network layer protocols and CAN frame processing



CAN_RAW – Reading and writing of raw CAN frames



- Similar to known programming interfaces
  - A socket feels like a private CAN interface
  - per-socket CAN identifier receive filtersets
  - Easy migration of existing CAN software
- Multiple applications can run independently
  - Network transparency through local echo of sent frames
  - Functions can be split into different processes
- Extra information via recvmsg systemcall
  - Linux timestamps in nano second resolution
  - Data flow information (internal/external origin, frame drops)

CAN_RAW – Example CAN-over-WLAN Bridge

Vehicle 1

Vehicle 2

WLAN-PCCard

CAN-PCCard

Wireless LAN
UDP/IP

WLAN-PCCard

CAN-PCCard

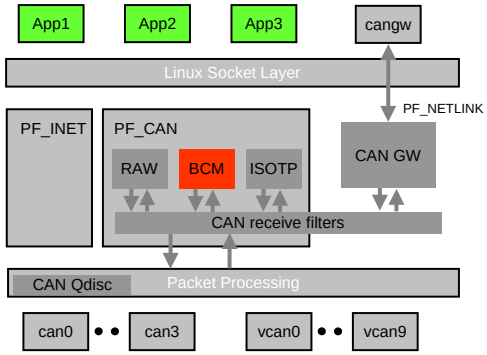
CAN

CAN

CAN_RAW – Example CAN-over-WLAN Bridge

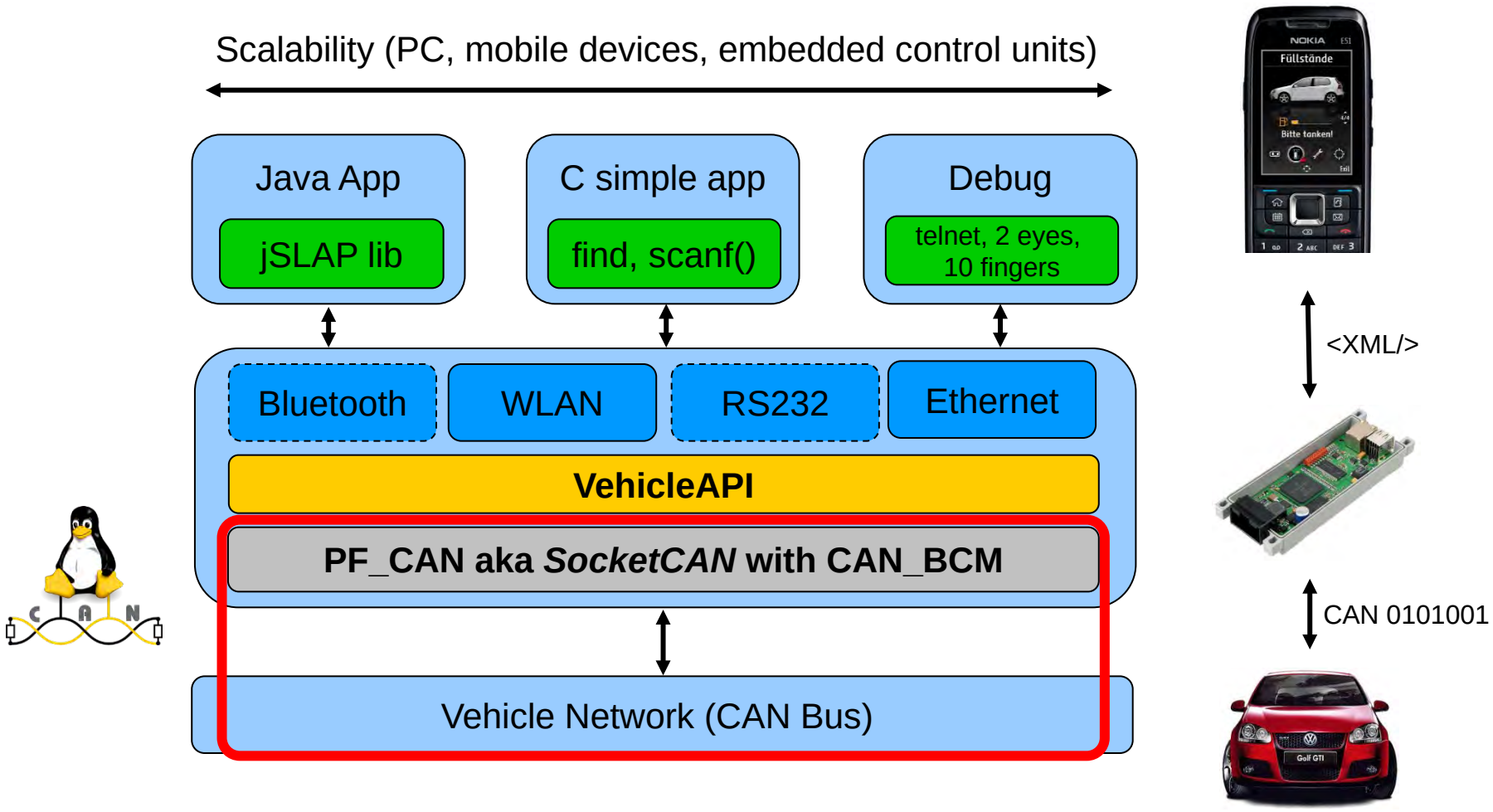
```
(..) /* some source code – don't worry */

int can;
int wlan;
struct can_frame mymsg;


can = socket(PF_CAN, SOCK_DGRAM, CAN_RAW);  /* CAN RAW Socket */
wlan = socket(PF_INET, SOCK_DGRAM, 0);      /* UDP/IP Socket  */

(..) /* set IP addresses, CAN interface and CAN filters */

bind(can, (struct sockaddr *)&can_addr, sizeof(can_addr));
connect(wlan, (struct sockaddr *)&in_addr, sizeof(in_addr));

while (1) {
           read(can, &mymsg, sizeof(struct can_frame));
           write(wlan, &mymsg, sizeof(struct can_frame));
}
```

CAN_BCM – timer support and filters for cyclic messages



- Executes in operating system context
- Based on Linux high resolution timers
- Covers OSEK automotive signal handling
- Programmable by BCM socket commands
- CAN receive path functions
  - Filter bit-wise content in CAN frame payload
  - Throttle update rate for changed received data
  - Detect timeouts of cyclic messages (deadline monitoring)
- CAN transmit path functions
  - Autonomous timer based sending of CAN frames
  - Multiplex CAN messages and instant data updates

CAN_BCM – Vehicle data access prototyping technology



CAN_BCM & VehicleAPI access for Car2Car applications

US Department of Transportation

- Vehicle Infrastructure Initiative project (DOT VII)

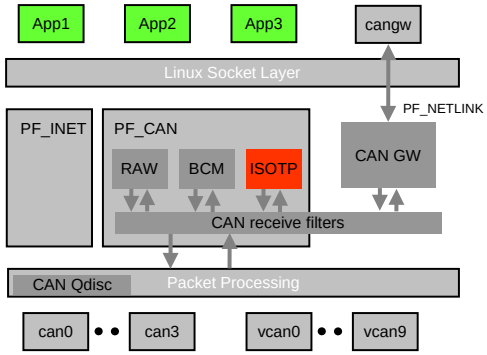
BMBF funded projects

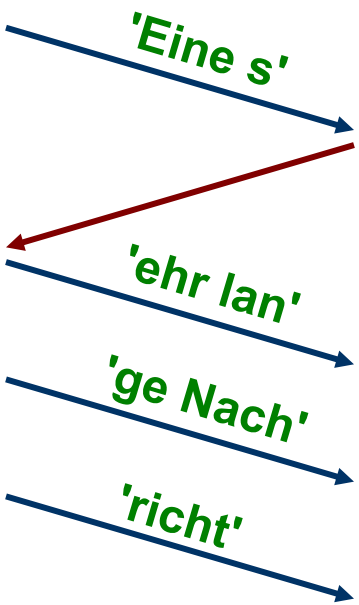
(federal german research)

- simTD
  Sichere Intelligente Mobilität
  Testfeld Deutschland
- Network-on-Wheels
  (BMW, Daimler, Opel, Audi)

CAN_ISOTP – CAN transport protocol ISO 15765-2

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





First Frame

Flow Control (stmin = 1 sec)

Consecutive Frame

Consecutive Frame

Consecutive Frame

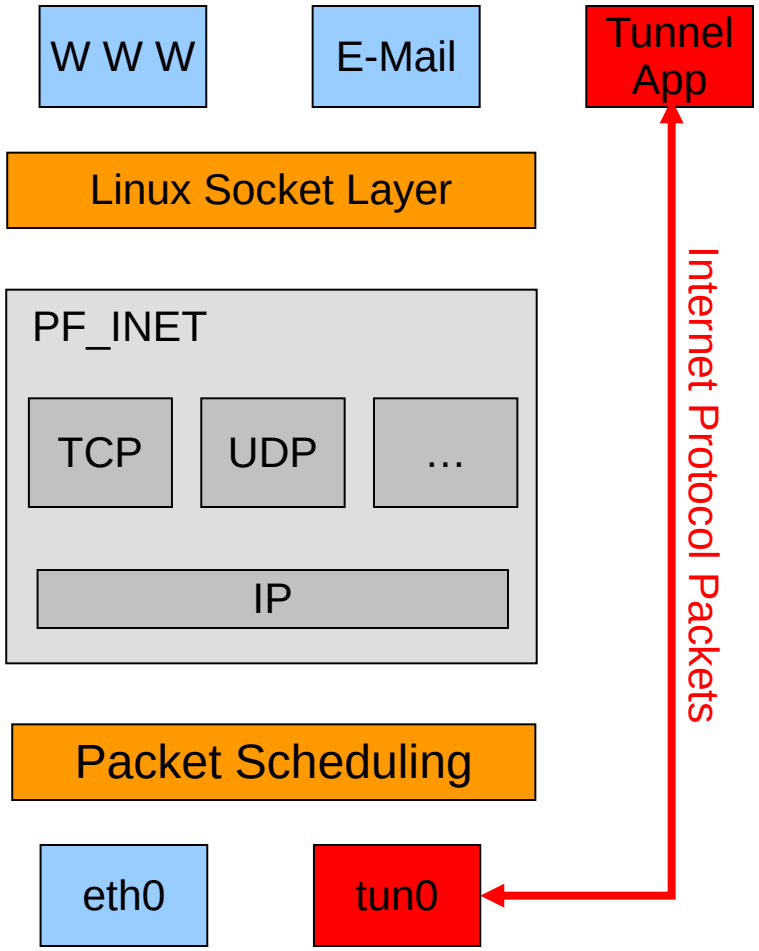
CAN_ISOTP – Example source code

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

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

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

addr.can_family = AF_CAN;                         /* address family AF_CAN */
addr.can_ifindex = ifr.ifr_ifindex;               /* CAN interface index e.g. 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 datagramm communication */

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

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

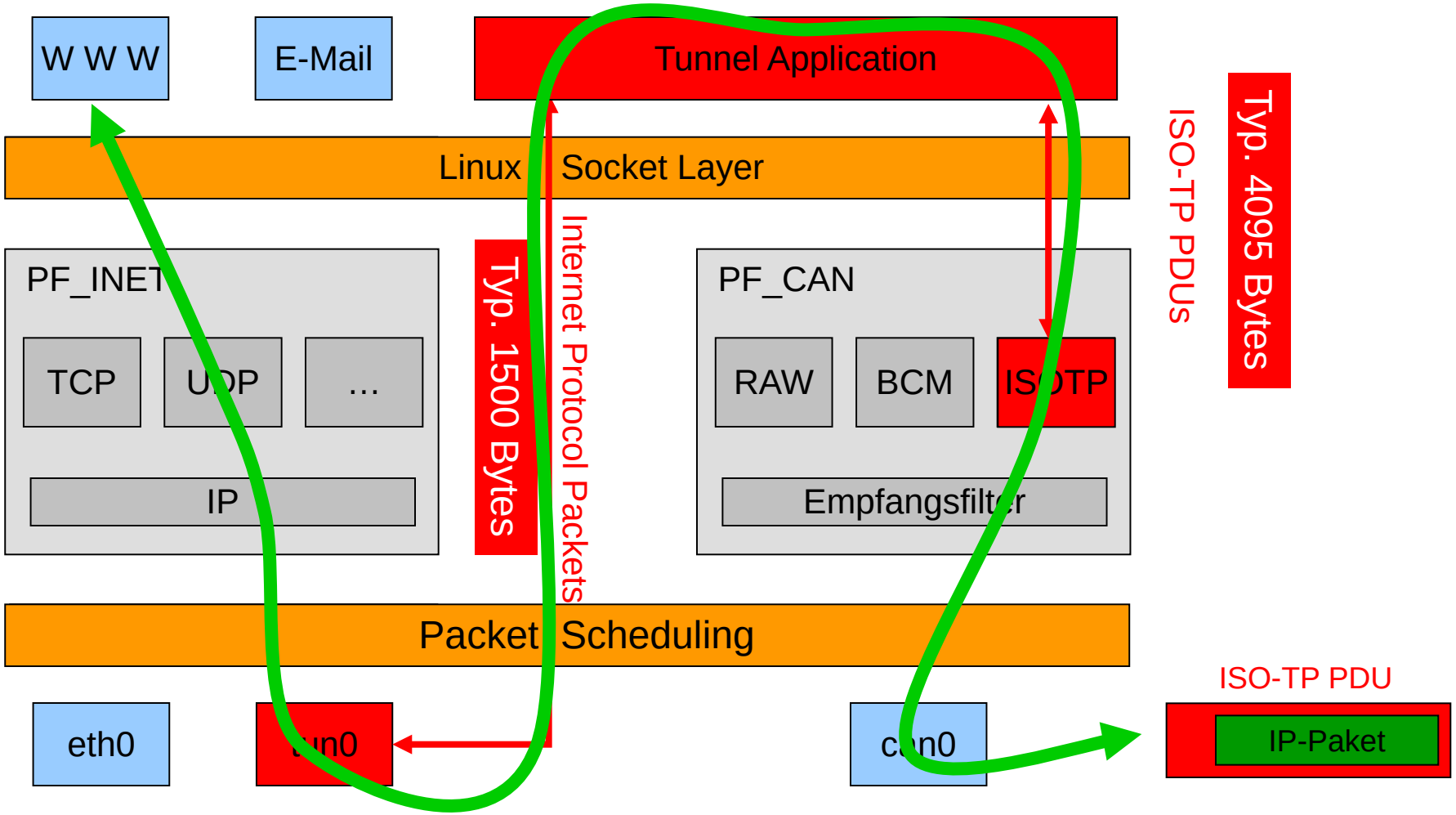
Archived the objective!
'Normal' application programmers can easily write applications for the vehicle using established techniques from the standard-IT!

IP over CAN: How to build an Internet Protocol Tunnel?

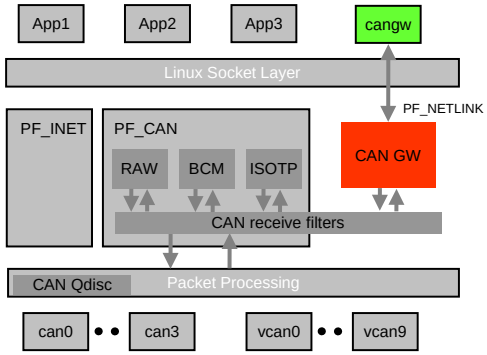



```
int t;
struct ifreq ifr;

t = open("/dev/net/tun", O_RDWR);

memset(&ifr, 0, sizeof(ifr));
ifr.ifr_flags = IFF_TUN | IFF_NO_PI;

strncpy(ifr.ifr_name, "tun%d", IFNAMSIZ);
ioctl(t, TUNSETIFF, (void *) &ifr);

/* now we have a tun0 (or tun1 or ...)   */
/* netdevice connected to filedescriptor t */
```

Internet Protokoll Tunnel over ISO TP 15765-2



CAN_GW – Linux kernel based CAN frame routing



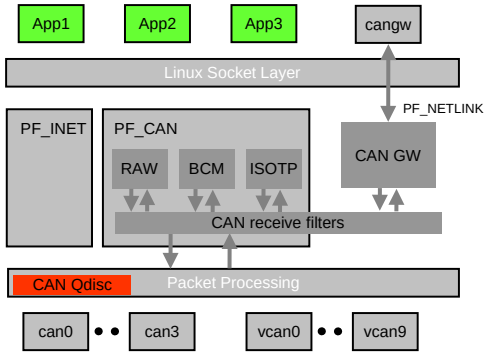
- Efficient one-hop CAN frame routing
- Re-use of Linux networking technology
  - PF_CAN receive filter capabilities
  - Linux packet processing NET_RX softirq
  - PF_NETLINK based configuration interface (known from Linux network routing configuration)
- Optional CAN frame modifications on the fly
  - Modify CAN identifier, data length code, payload data with AND/OR/XOR/SET operations
  - Calculate XOR and CRC8 checksums after modification
  - Support of different CRC8 profiles (1U8, 16U8, SFFID_XOR)

CAN Gateway userspace tool for netlink configuration

```
Usage: cangw [options]

Commands:   -A (add a new rule)
            -D (delete a rule)
            -F (flush / delete all rules)
            -L (list all rules)

Mandatory: -s <src_dev>  (source netdevice)
           -d <dst_dev>  (destination netdevice)
Options:   -t (preserve src_dev rx timestamp)
           -e (echo sent frames - recommended on vcanx)
           -f <filter> (set CAN filter)
           -m <mod> (set frame modifications)
           -x <from_idx>:<to_idx>:<result_idx>:<init_xor_val> (XOR checksum)
           -c <from>:<to>:<result>:<init_val>:<xor_val>:<crctab[256]> (CRC8 cs)
           -p <profile>:[<profile_data>] (CRC8 checksum profile & parameters)

Values are given and expected in hexadecimal values. Leading 0s can be omitted.

<filter> is a <value>:<mask> CAN identifier filter

<mod> is a CAN frame modification instruction consisting of
<instruction>:<can_frame-elements>:<can_id>.<can_dlc>.<can_data>
 - <instruction> is one of 'AND' 'OR' 'XOR' 'SET'
 - <can_frame-elements> is _one_ or _more_ of 'I'dentifier 'L'ength 'D'ata
 - <can_id> is an u32 value containing the CAN Identifier
 - <can_dlc> is an u8 value containing the data length code (0 .. 8)
 - <can_data> is always eight(!) u8 values containing the CAN frames data
The max. four modifications are performed in the order AND -> OR -> XOR -> SET

Example:
cangw -A -s can0 -d vcan3 -e -f 123:C00007FF -m SET:IL:333.4.1122334455667788

Supported CRC 8 profiles:
Profile '1' (1U8)       - add one additional u8 value
Profile '2' (16U8)      - add u8 value from table[16] indexed by (data[1] & 0xF)
Profile '3' (SFFID_XOR) - add u8 value (can_id & 0xFF) ^ (can_id >> 8 & 0xFF)
```

Traffic shaping for CAN frames

- Multiple applications can share one CAN bus
- Different per-application requirements
  - Timing requirements for cyclic messages or transport protocol timeouts
  - Bandwidth requirements
  - Ensure priority handling for outgoing CAN frames (CAN network interfaces implement a short FIFO queue)
- Similar requirements are known from Internet Protocol traffic (e.g. to reduce bandwidth for peer-to-peer networking)



Traffic shaping for CAN frames – Example

App1

Send status information every 200ms

App2

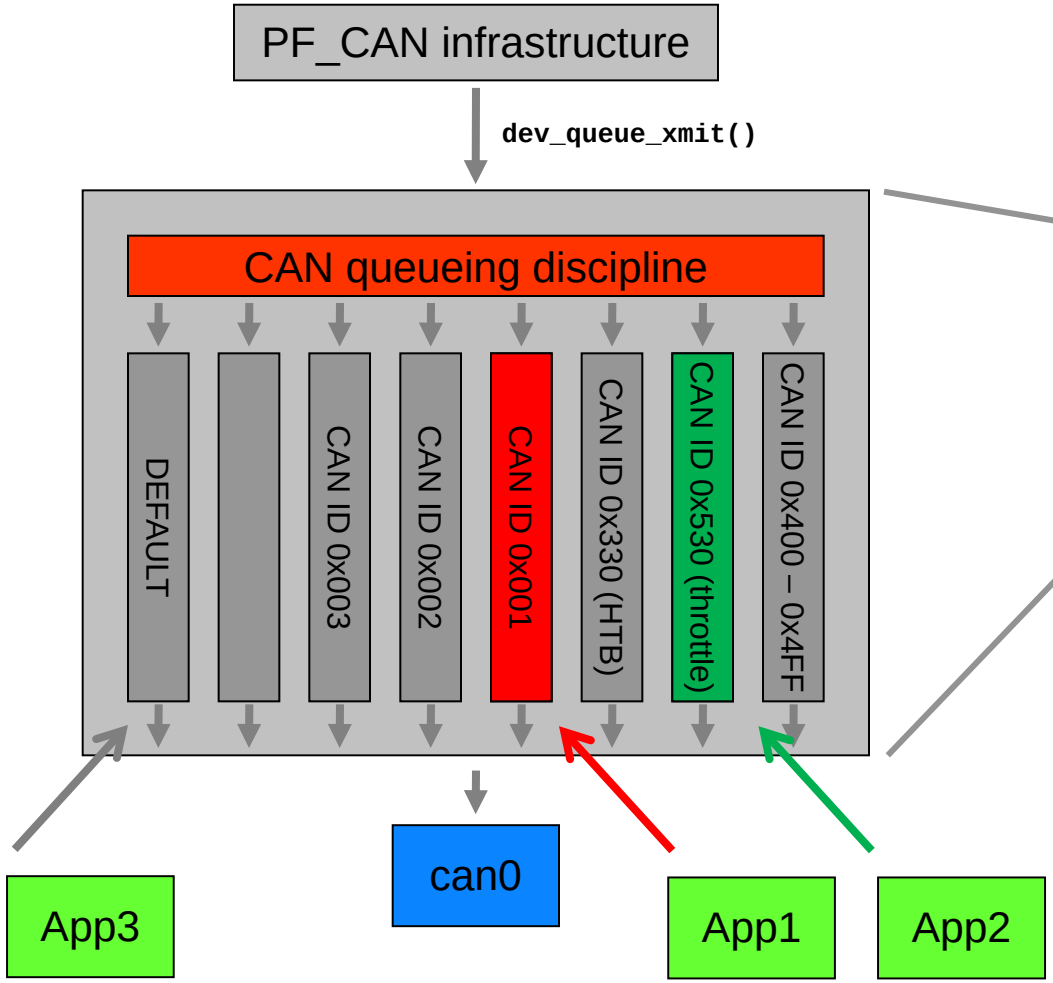
Send bulk data, like a ISO TP PDU with stmin = 0 (no delay)

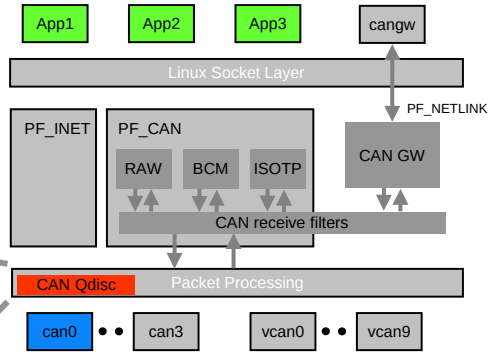
Traffic shaping for CAN frames – Example

FIFO policy

- Timeouts
- Outdated data

Linux Queueing Disciplines for CAN Frames



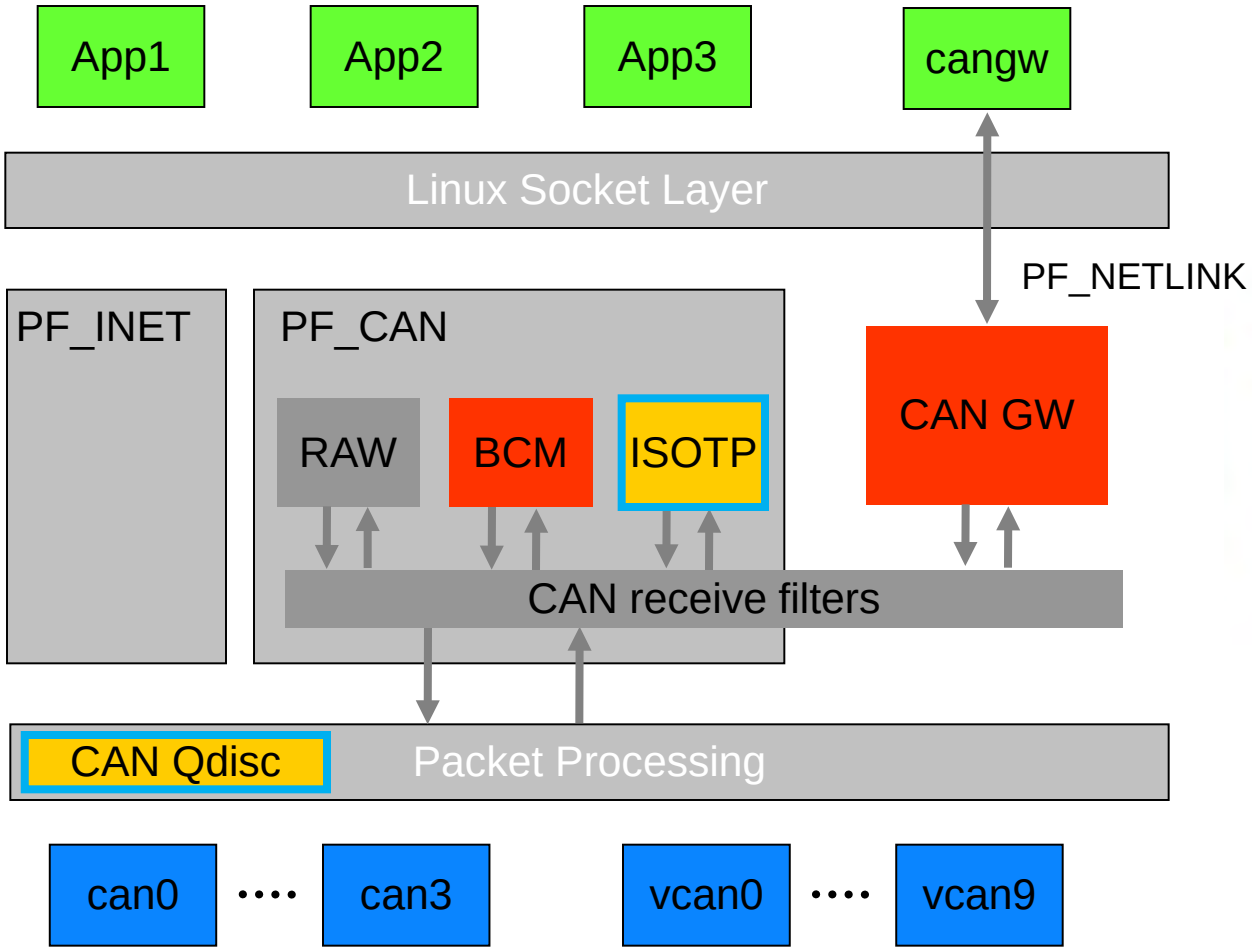


- Implement a new packet scheduler classifier cls_can
- Extend the traffic control tool ***tc*** to support CAN IDs

Traffic shaping for CAN frames – Example

Application specific
Qdisc configuration
(by host admin / root)

Summary



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

$> _
```