

eBPF on Wheels

Automotive and Industrial Use Cases

Linux Plumbers 2026, Reinhard Kugler



 Bundesministerium
Klimaschutz, Umwelt,
Energie, Mobilität,
Innovation und Technologie

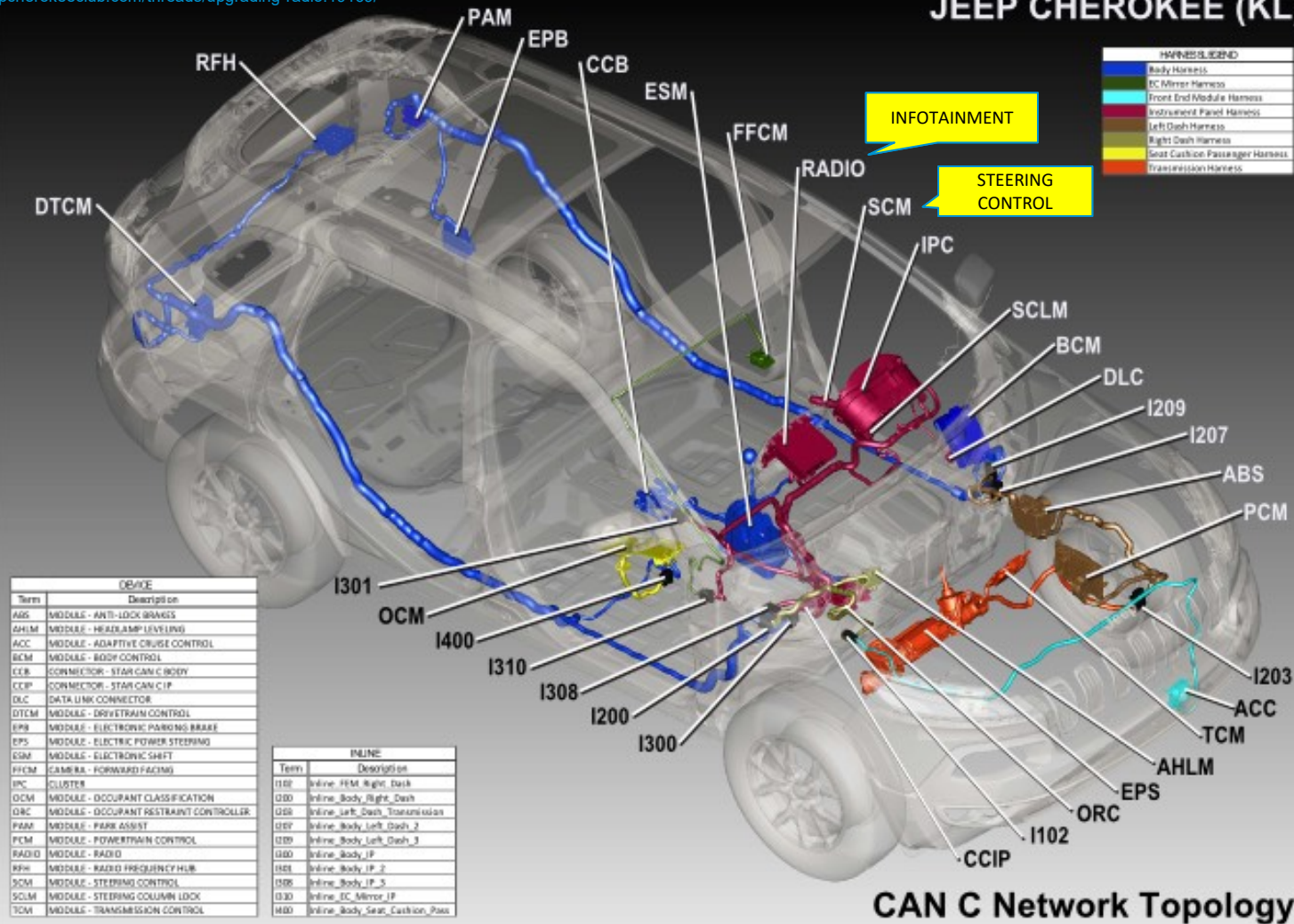
 Bundesministerium
Arbeit und Wirtschaft



FWF Österreichischer
Wissenschaftsfonds



JEEP CHEROKEE (KL)



E/E architectures

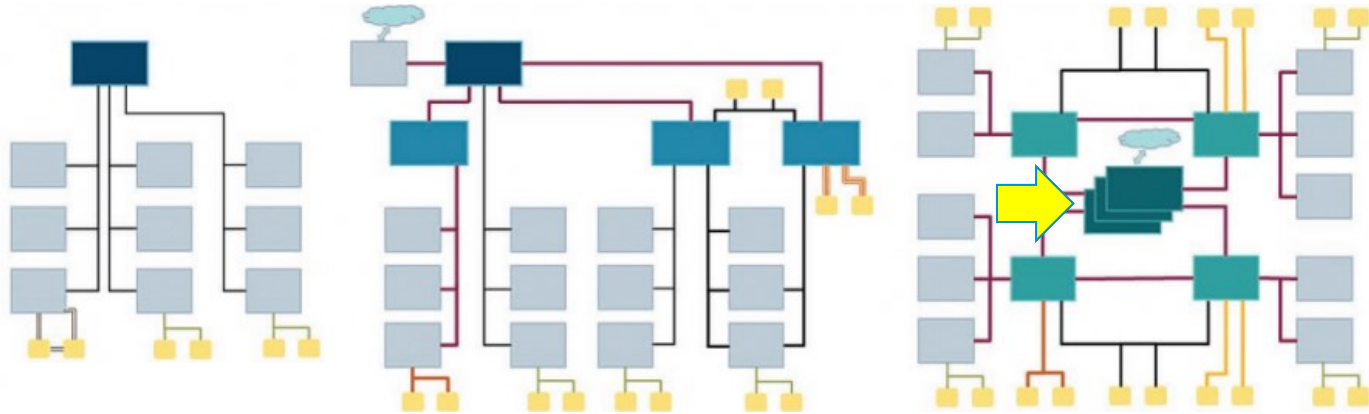


Image: <https://www.techdesignforums.com/blog/2021/11/17/balancing-the-requirements-of-e-e-architectures-for-automotive-design/>

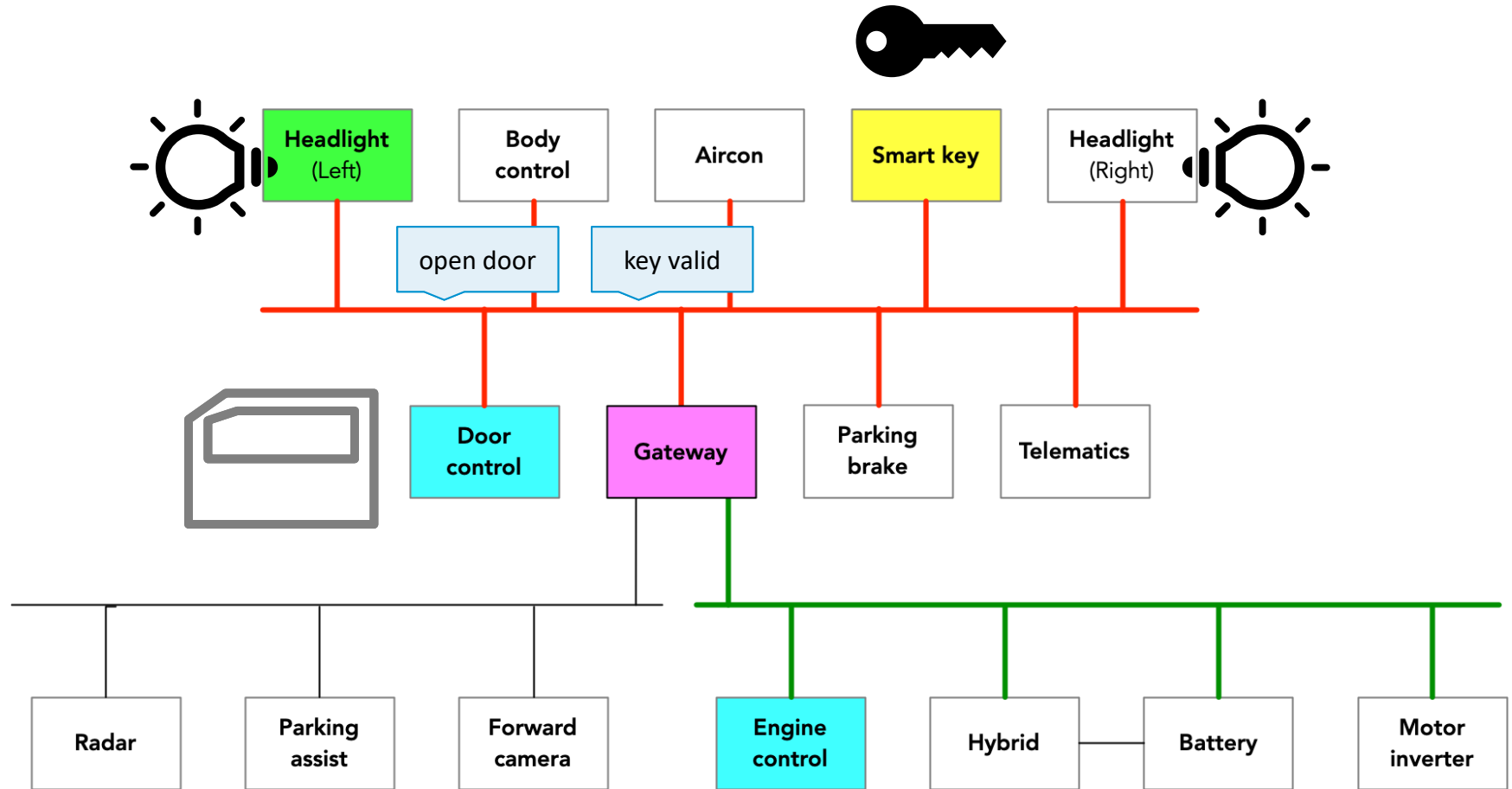
dedicated ECUs
multi-bus + gateway

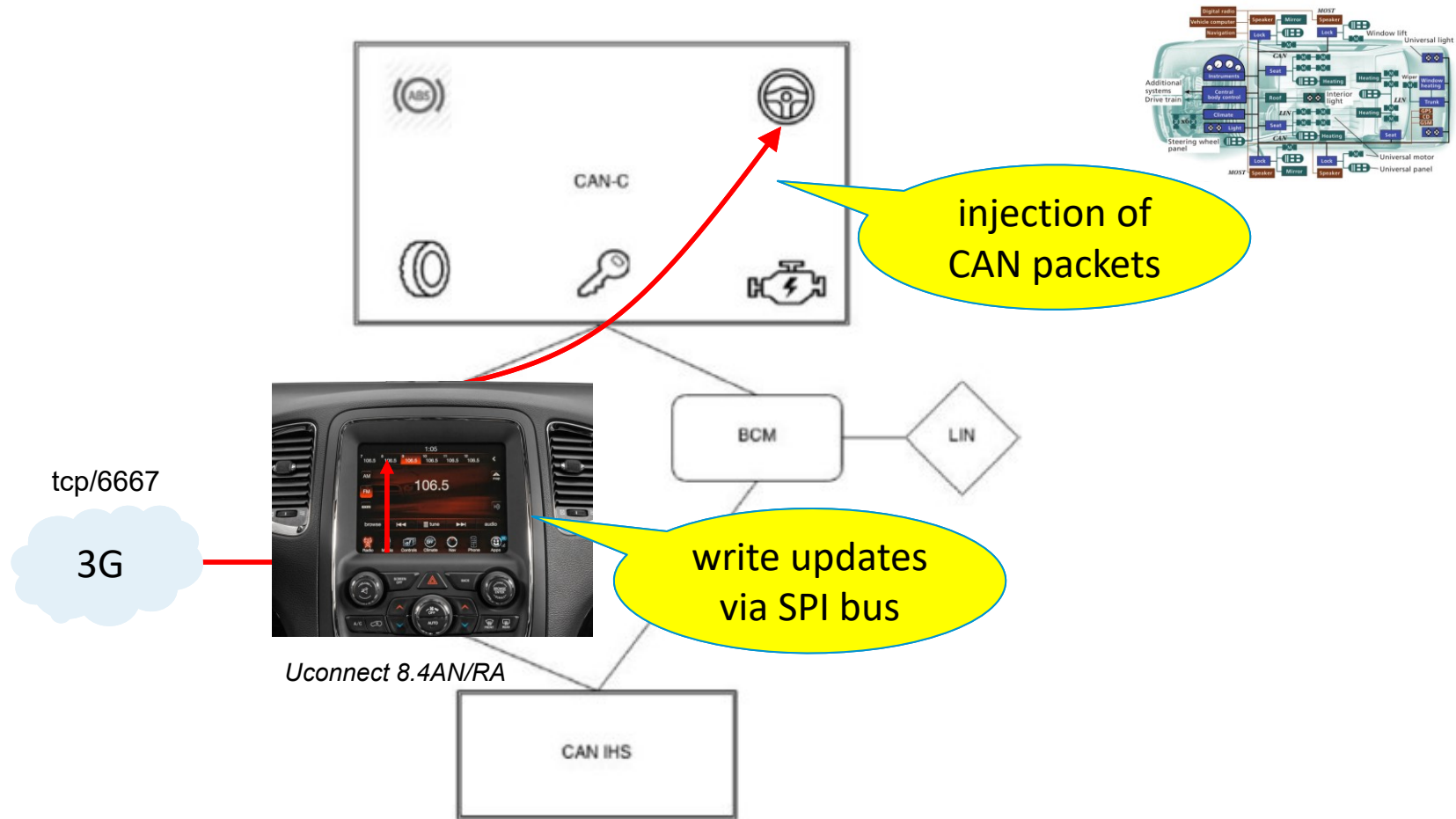


domain controller
architecture



zonal architecture,
centralized compute



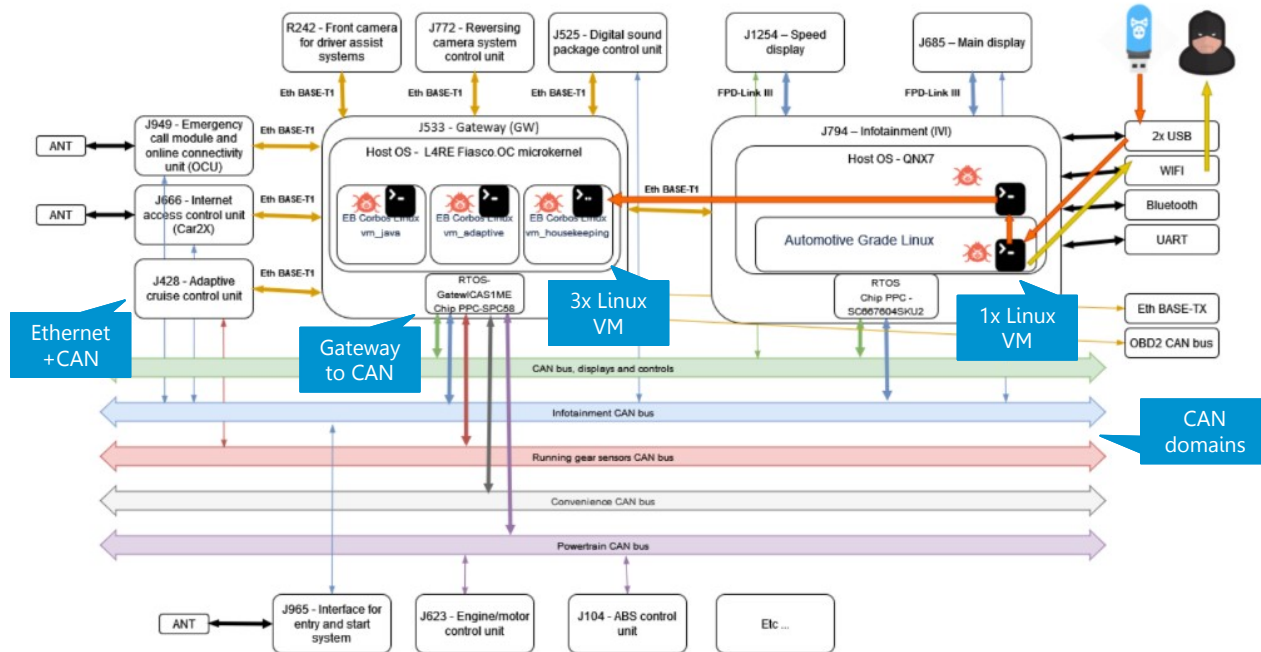




110.56KB/s

<https://www.youtube.com/watch?v=3Wd4H91qfso>

Attack vector via USB interfaces



Running containers in cars

October 19, 2022 | Pierre-Yves Chibon, Daniel J Walsh, Alexander Larsson

< Back to all posts

Tags: Co

A little over a year ago, Red Hat [announced](#) its intention to collaborate with the automotive industry to help drive the transition to software-defined vehicles (SDVs). In May 2022, this intention became concrete with Red Hat and General Motors [announcing their collaboration](#) to help trailblaze SDVs at the edge. Our goal is to produce a base operating system to run all sorts of in-vehicle software for safety critical use cases as well as non-safety ones.

Containers have become the de facto standard in the wider IT industry, and they're front and center in the vision of the [software-defined vehicle](#), allowing for applications to be isolated, providing more flexibility for developers and deployment, and generally allowing for faster innovation. Red Hat leads the work on containers in the cloud, and now it's time to take that expertise to the automotive industry.

Smart Cockpit HPC

Continental's Smart Cockpit High-Performance Computer (HPC) targets the sweet spot between user experience, costs, and system performance, while enabling a short development time within only 18 months. It combines the powerful processors of the Telechips Dolphin family with pre-integrated functions of cluster, infotainment and ADAS visualization. In collaboration with Google Cloud, Continental integrates generative AI technology directly into the vehicle server. This allows drivers to interact with their car in a natural dialogue.



Open-source operating system for HPC

EB <https://www.bellat.com/en/blog/running-containers-cars>

<https://www.continental-automotive.com/en/components/hpc-product-portfolio.html>

<https://www.elektrobit.com/products/ecu/eb-corbos/linux-built-on-ubuntu/>

Contribution and Request for Comment



I would like to show:

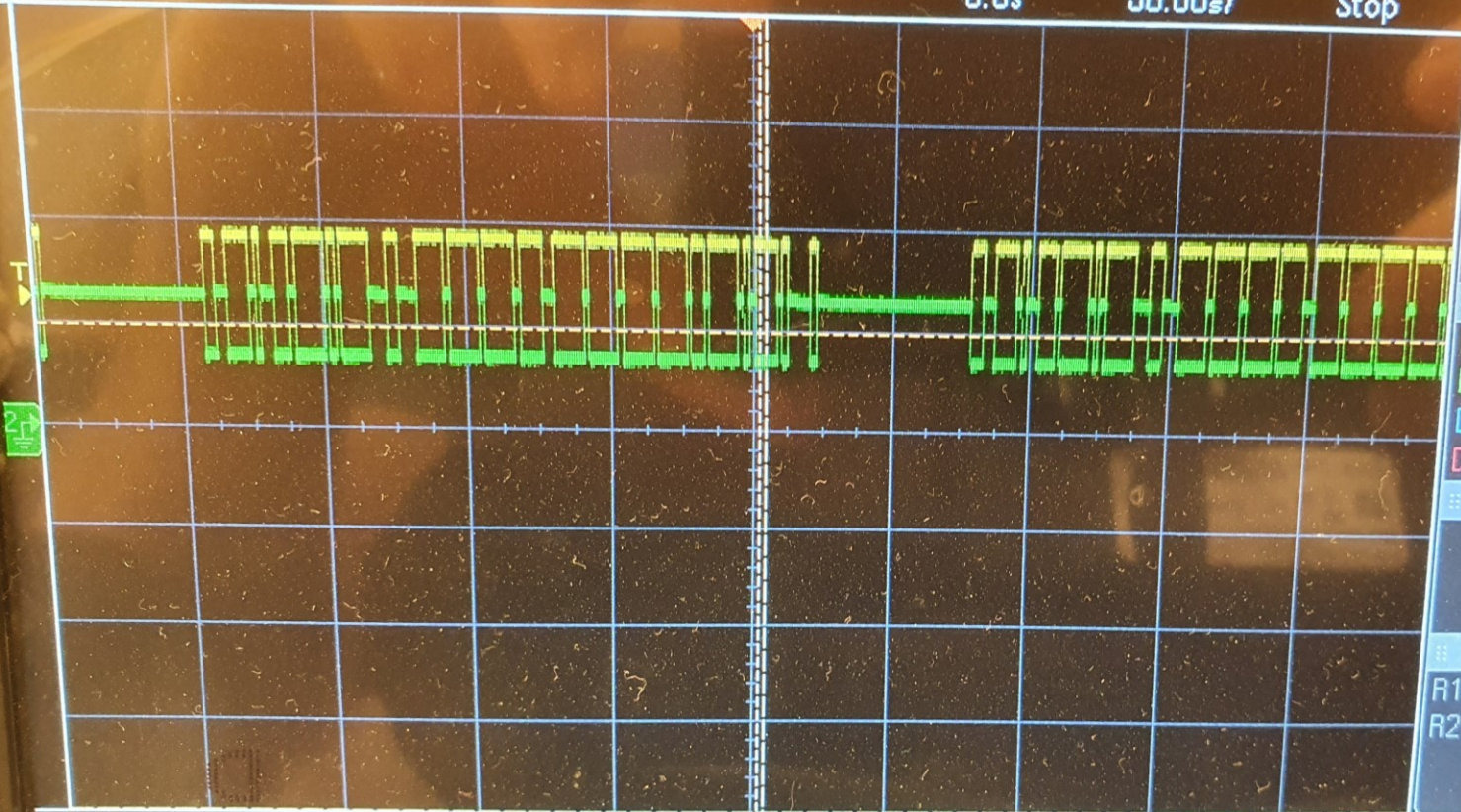
- the applied use of eBPF for cybersecurity aspects in CAN and SPI (edge cases)

I kindly ask for your feedback:

- on variables in measurements
- network integration aspects



1 2.00V/ 2 2.00V/ 3 4 0.0s 50.00%/ Stop 1 2.48V



KEYSIGHT
TECHNOLOGIES

Acquisition
Normal
500MSa/s

Channels
DC 10.0:1
DC 10.0:1
DC 10.0:1
DC 10.0:1

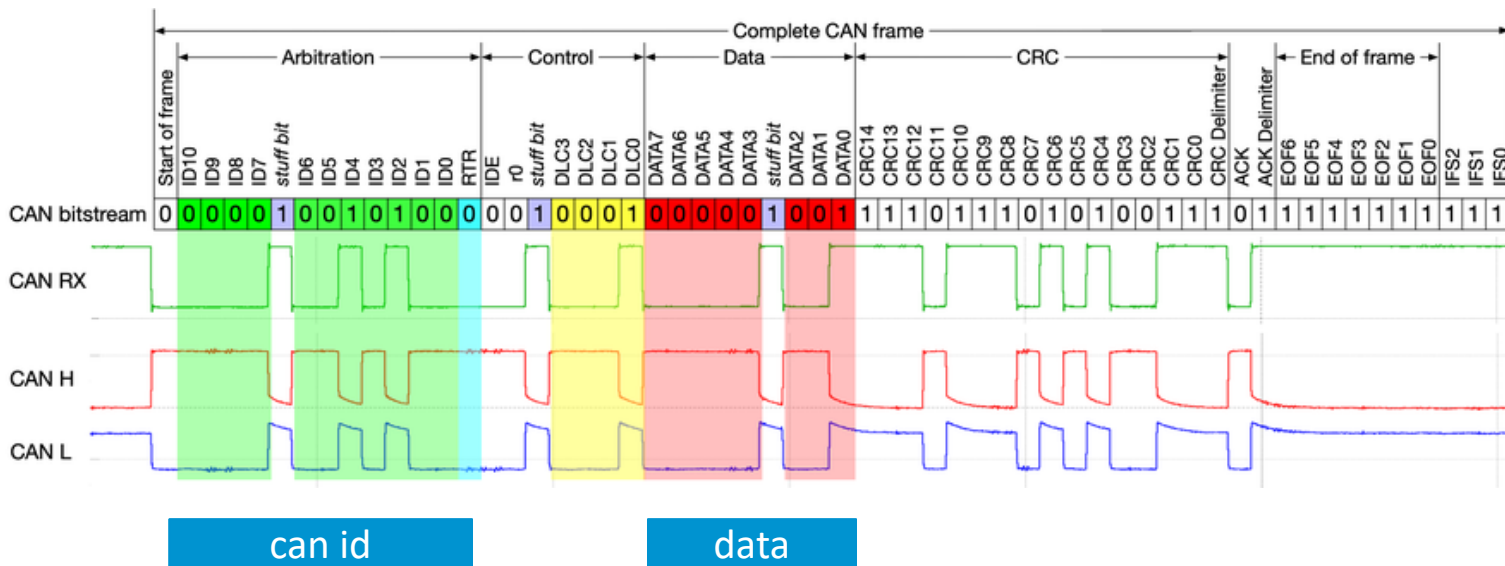
Math
Ch1 + Ch2
1.00V/
31.3mV

Ref
R1: 5.00V/ 0.0V
R2: 5.00V/ 0.0V

ΔX : +526.3157895kHz
 $\Delta Y(1)$: -9.96975V

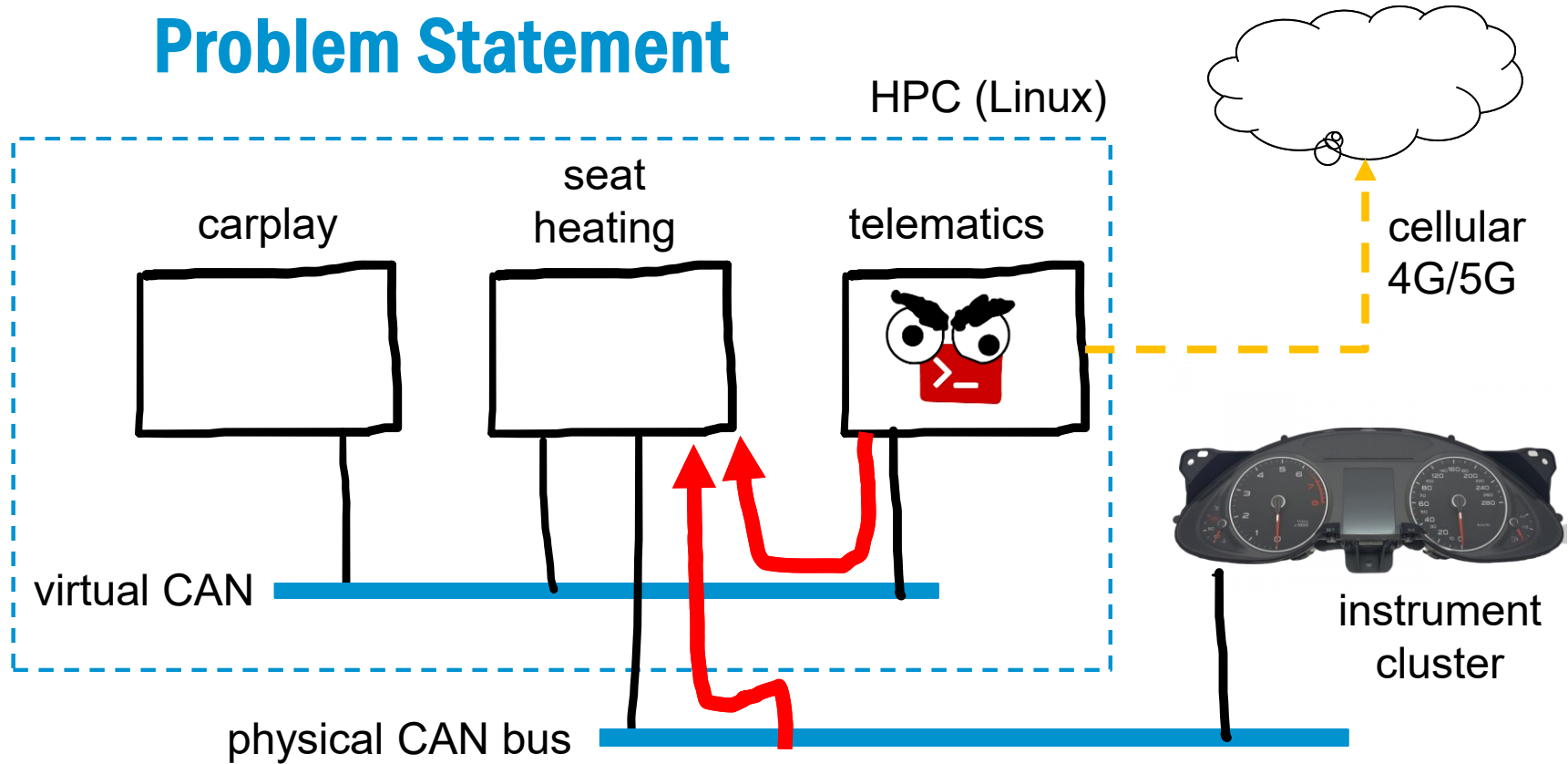
08:53 AM
Aug 22, 2022

CAN frame



Different variants: CAN CC (2.0), CAN FD, CAN XL
Alternatives: Automotive Ethernet, FlexRay, LIN, ...

Problem Statement



CAN in the Kernel and Networking

- CAN drivers for devices (built upon can-dev)
 - drivers/net/can/ (mcp251x, flexcan, peak_can)
- Virtual CAN interfaces (vcan and vxcan)
- CAN frames supported by the Linux Networking Subsystem (dealing with Socket Buffer)
- socket-can acts as an interface in user space

CAN Frames of CAN FD and CANXL

```
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)));
};

struct canxl_frame {
    canid_t prio; /* 11 bit priority for arbitration / 8 bit VCID */
    __u8 flags; /* additional flags for CAN XL */
    __u8 sdt; /* SDU (service data unit) type */
    __u16 len; /* frame payload length in byte */
    __u32 af; /* acceptance field */
    __u8 data[CANXL_MAX_DLEN];
};
```

<https://github.com/linux-can/can-utils/blob/master/include/linux/can.h>

Attachment Points for eBPF on the CAN Bus

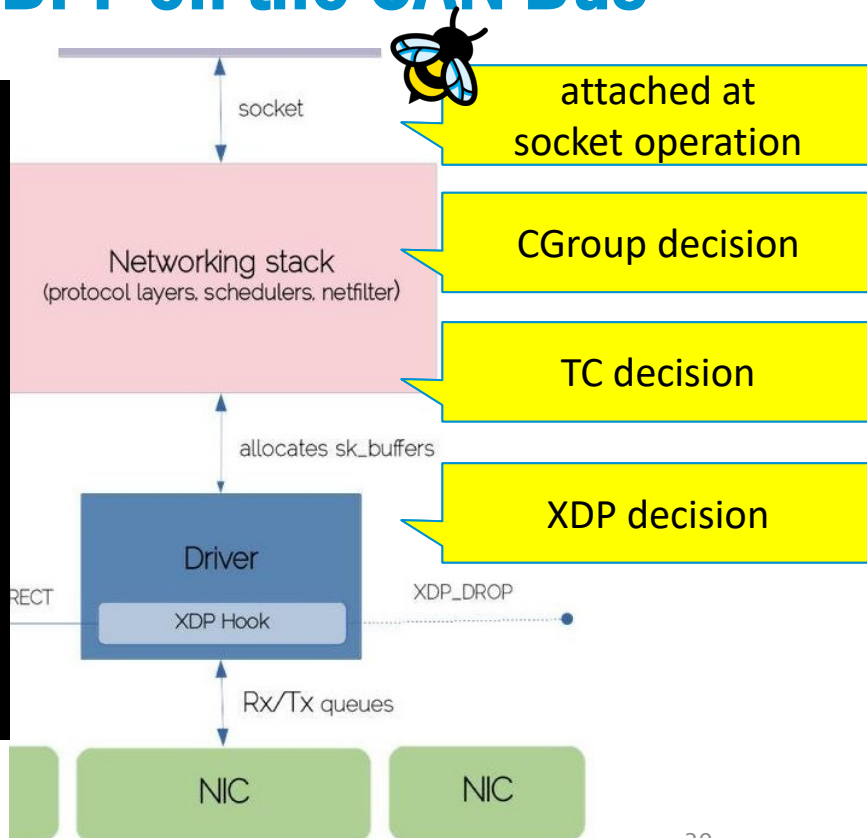
```
$ ip link show
```

```
3: can0: <NOARP,UP,LOWER_UP,ECHO> mtu 16 xdpgeneric  
qdisc pfifo_fast state UP mode DEFAULT group  
default qlen 10  
    link/can  
    prog/xdp id 400
```

```
18: vcan0: <NOARP,UP,LOWER_UP> mtu 72 qdisc noqueue  
state UNKNOWN mode DEFAULT group default qlen 1000  
    link/can
```

```
$ tc qdisc show dev can0  
qdisc pfifo_fast 0: root refcnt 2 bands 3 priomap 1  
2 2 2 1 2 0 0 1 1 1 1 1 1 1 1
```

```
$ tc qdisc show dev vcan0  
qdisc noqueue 0: root refcnt 2
```



```

struct can_frame {
    __u32  can_id; /* 32 bit CAN_ID + EFF/RTR/ERR flags */
    __u8   can_dlc; /* data length code: 0 .. 8 */
    __u8   data[8] __attribute__((aligned(8)));
};

SEC("xdp")
int xdp_can_drop_id0x123(struct xdp_md *ctx)
{
    void *data_end = (void *) (long) ctx->data_end;
    void *data = (void *) (long) ctx->data;
    struct can_frame *frame = data;
    if(frame->can_id == 0x123) // drop frames with CAN ID 0x123
packets
    {
        // bpf_printk("dropping CAN frame with ID 0x123.");
        return XDP_DROP;
    }

    // default action

```

eBPF Filter for CAN frames

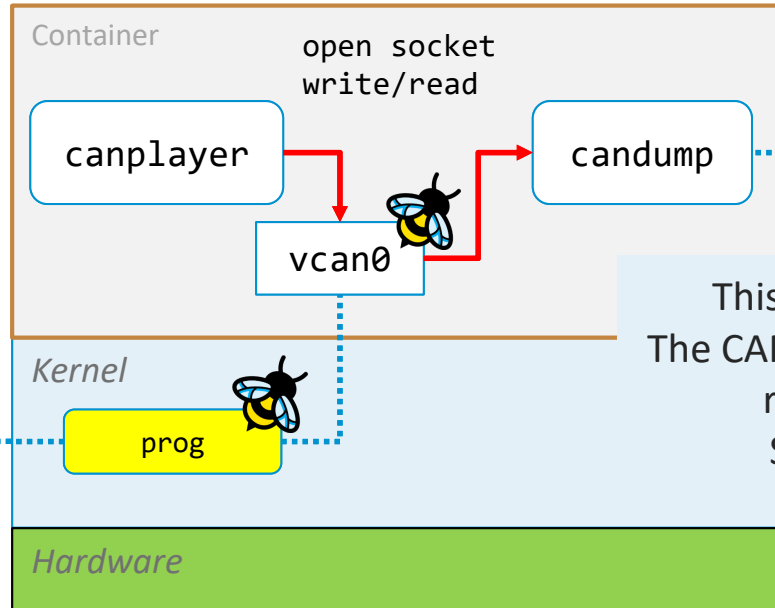
- XDP attached eBPF program to filter CAN frames
- acts as simple CAN firewall
- extended version can use maps for IDs and operation modes
- compared to existing bit-masks we can create more complex checks.

Experiment 1: XDP filter for CAN CC (2.0)

Used CAN traffic from a capture-file and send it into a virtual interface and received the CAN frames with candump.

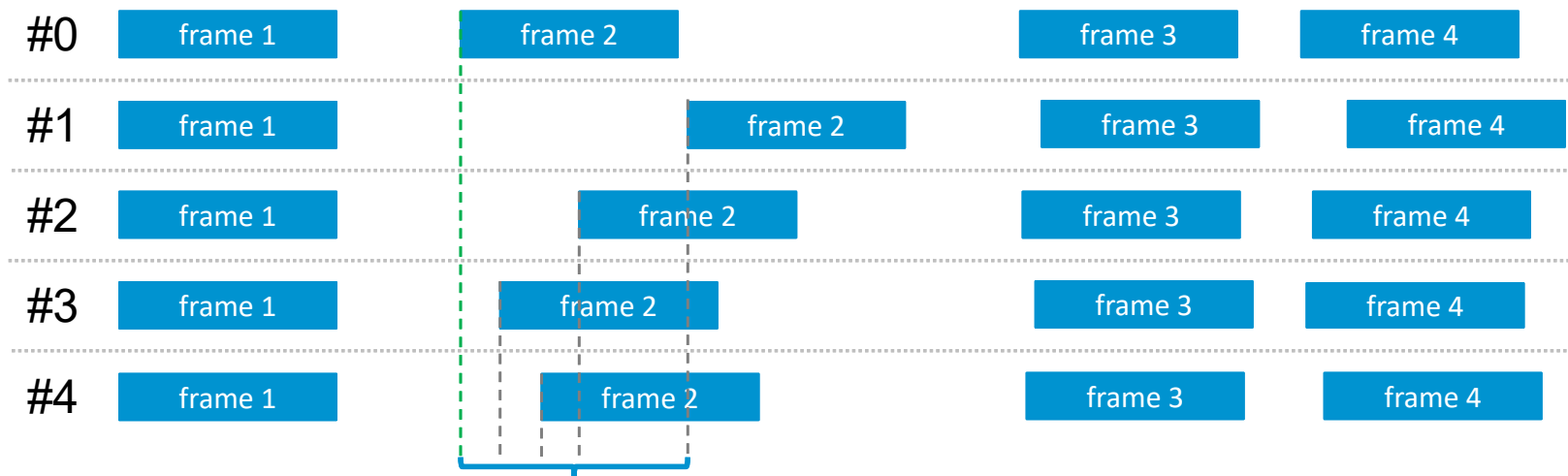
uses a capture file of a real car and replays it using the existing timestamps

dissects the CAN protocol and filters



Measurements

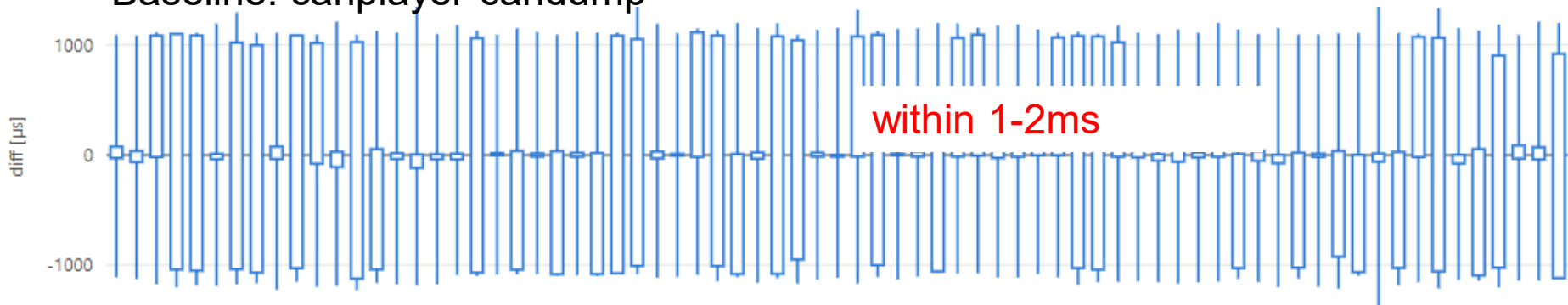
The measurements use a pre-recorded CAN capture of a real car and replays the 7223 CAN frames using stored timestamps each run.



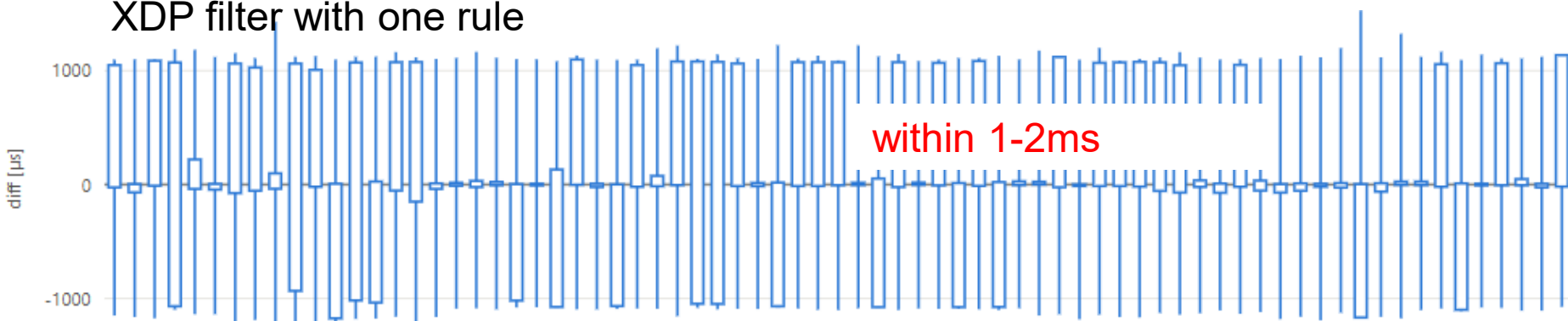
We measure delays in
the receiving the frames

CAN CC 0.5Mbps

Baseline: canplayer-candump

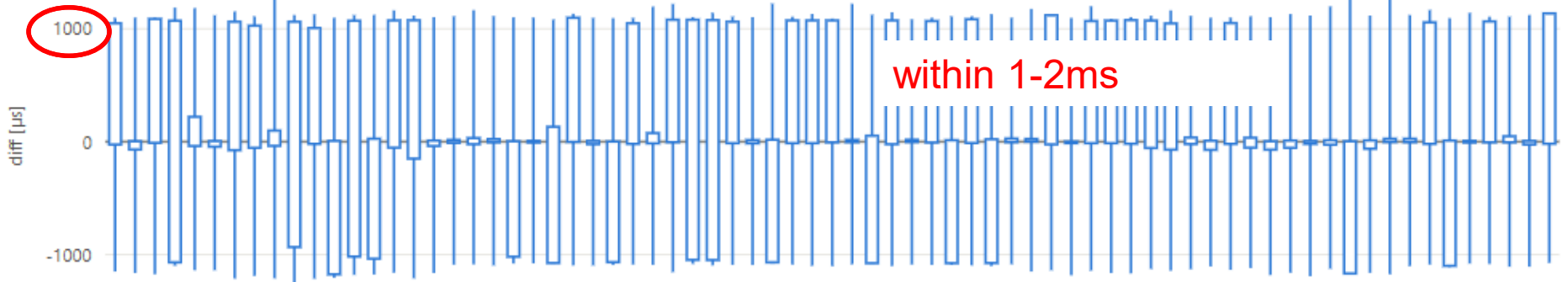


XDP filter with one rule



note, that the static diagrams were produced with insufficient data.

XDP filter on CAN with one rule



XDP filter with 1000 randomly generated ID-check rules

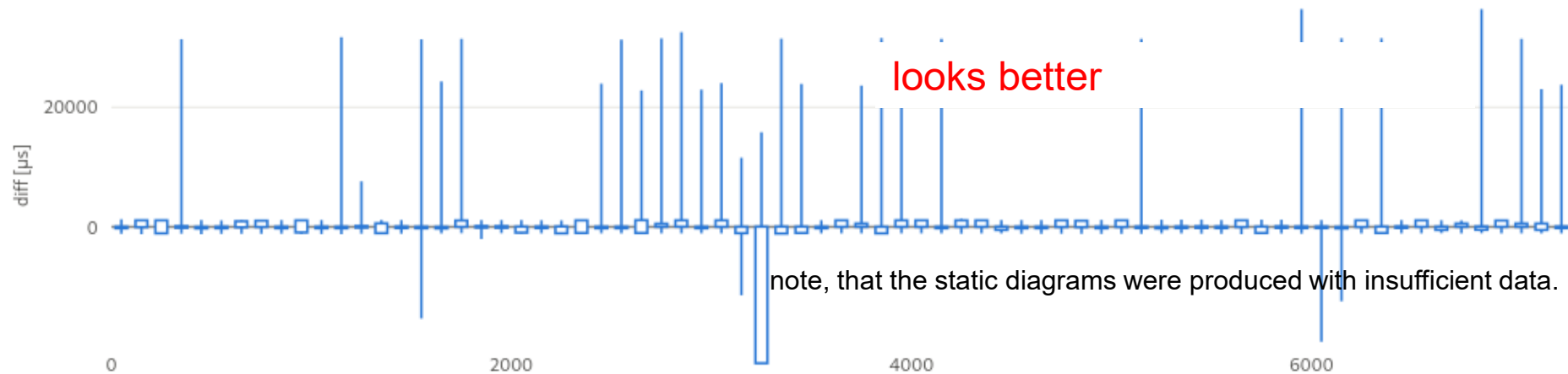


note, that the static diagrams were produced with insufficient data.

XDP filter with 1000 randomly generated ID-check rules



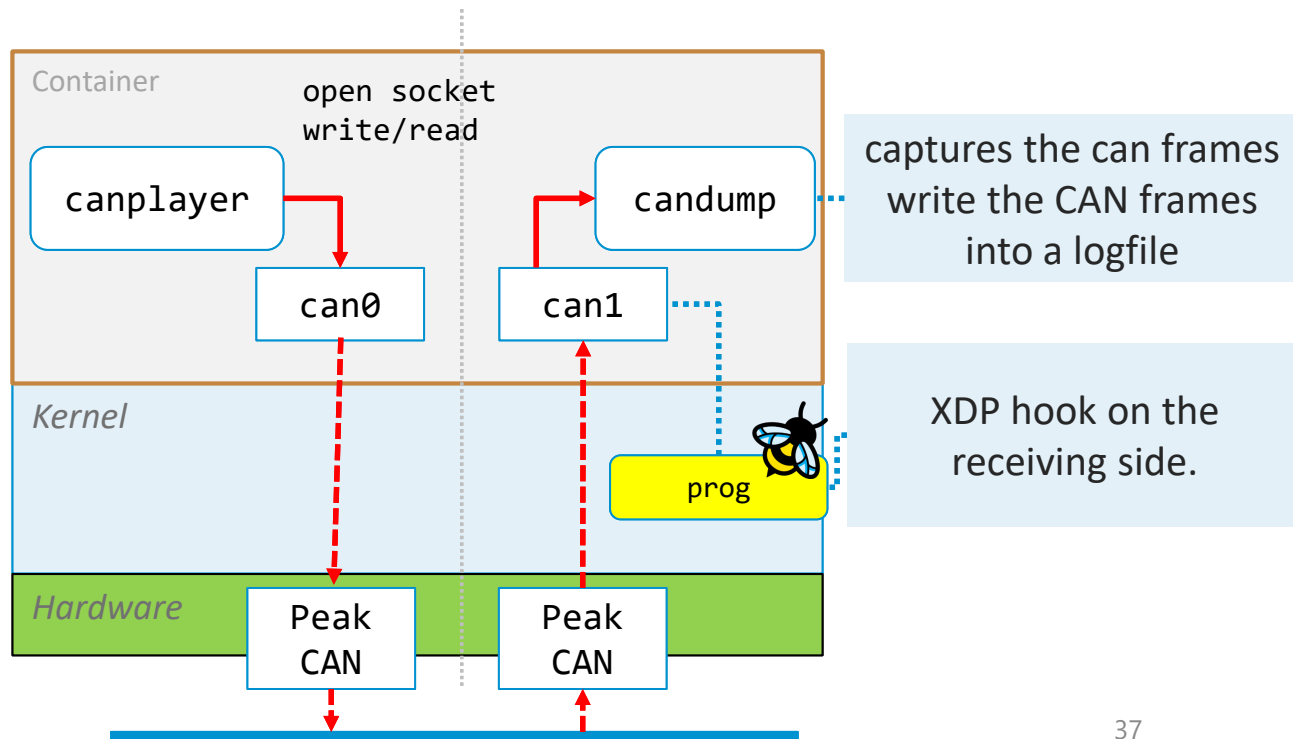
Same test (above) with net.core.bpf_jit_enable=0



Experiment 3: Hardware Setup (same device)

uses a capture file of a real car and replays it using the existing timestamps

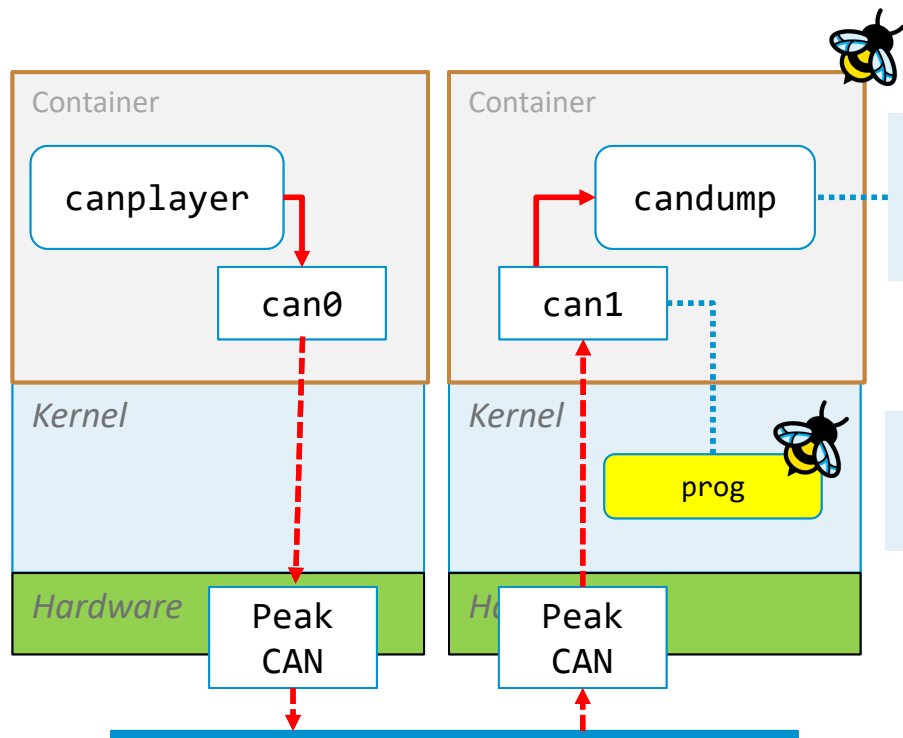
Professional CAN hardware and a 120 Ohm terminated bus



Experiment 4: Device to Device

uses a capture file of a real car and replays it using the existing timestamps

Professional CAN hardware and a 120 Ohm terminated bus



captures the can frames
write the CAN frames
into a logfile

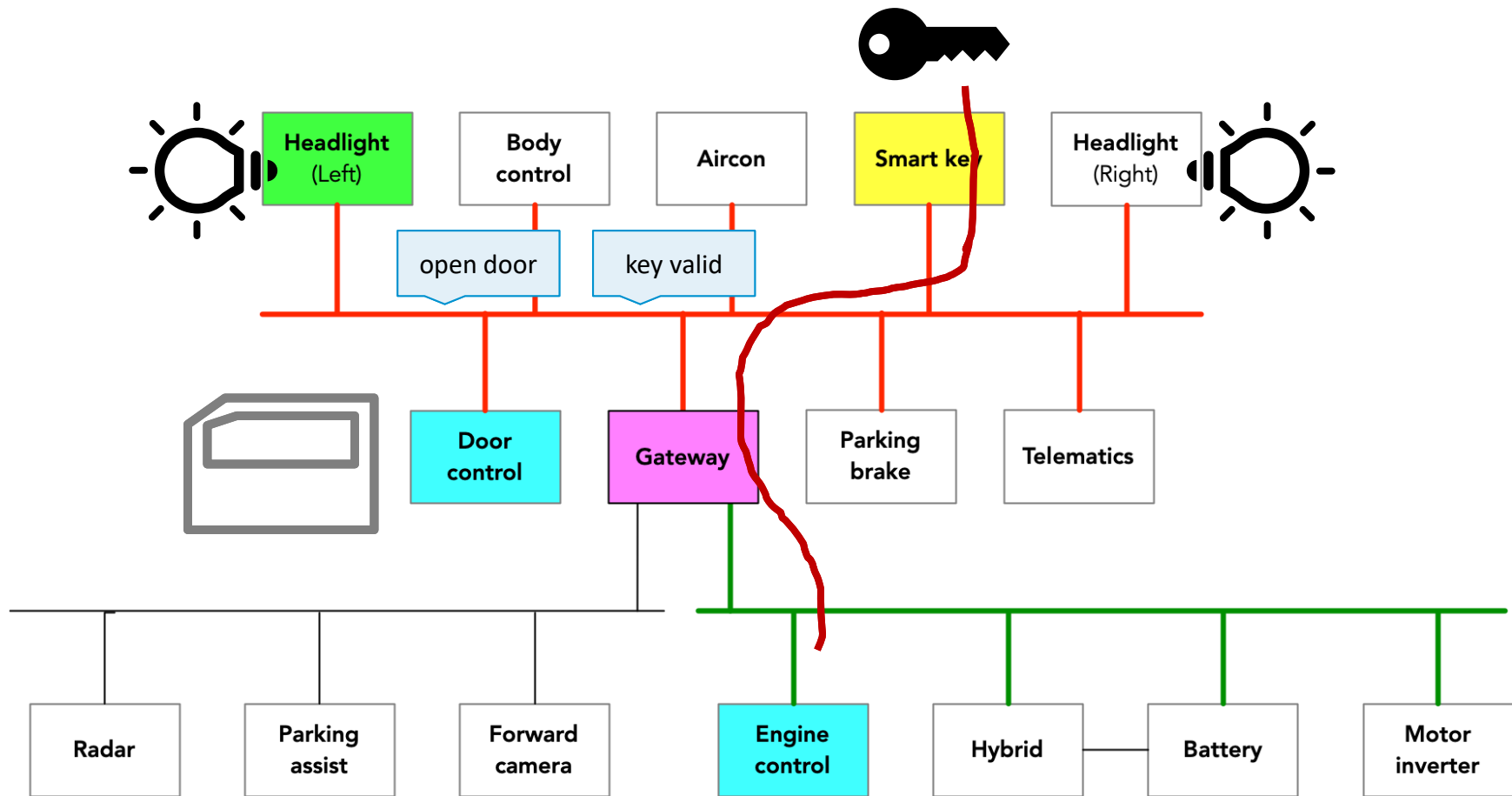
XDP hook on the
receiving side.

Discussion and RFC

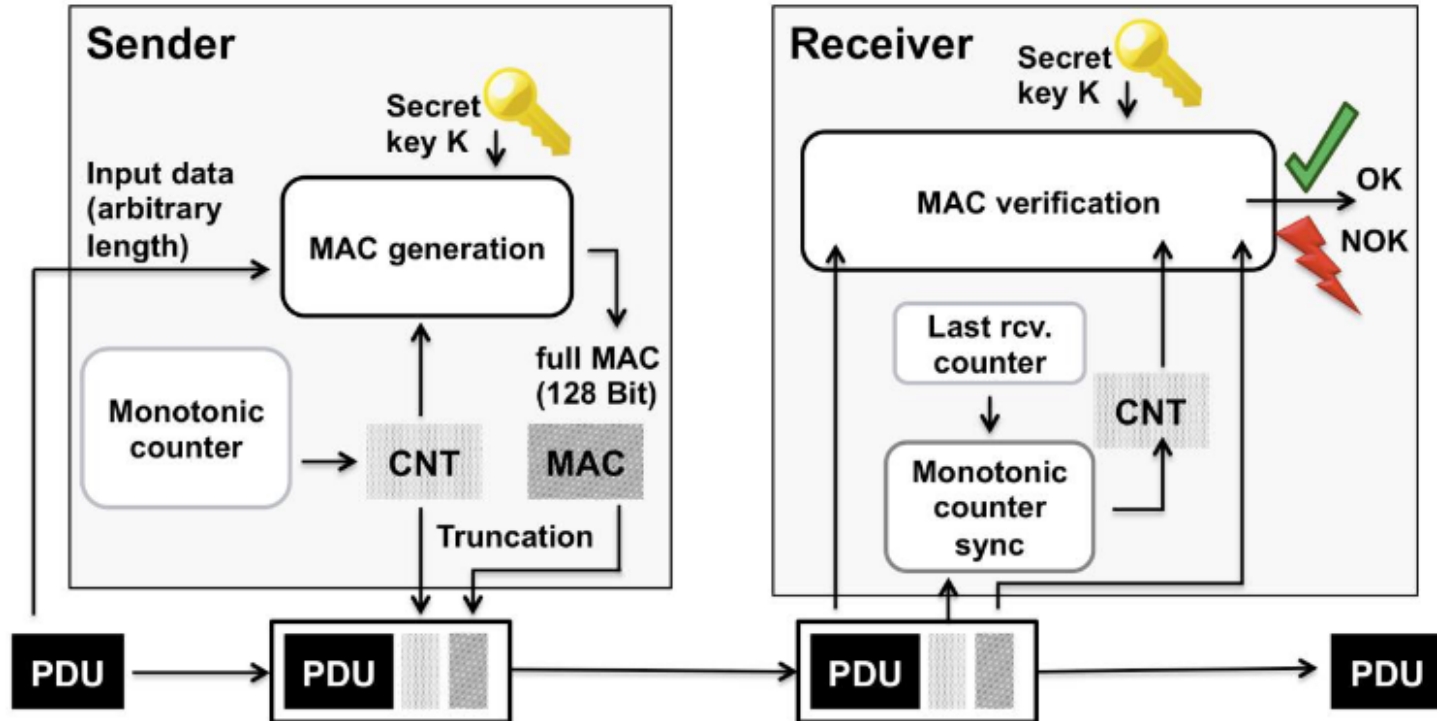
- Does the performance differ between XDP and TC (or socket)?
- Does the performance differ in processing of CAN CC, CAN FD, CAN XL?
 - we expect that the CAN frame fits in the XDP context
- What coding constructs introduce the delays?
- Is there a better way to measure/trace?
 - we measured received time between frames in userspace
 - ftrace?

User Space Helpers for eBPF on CAN

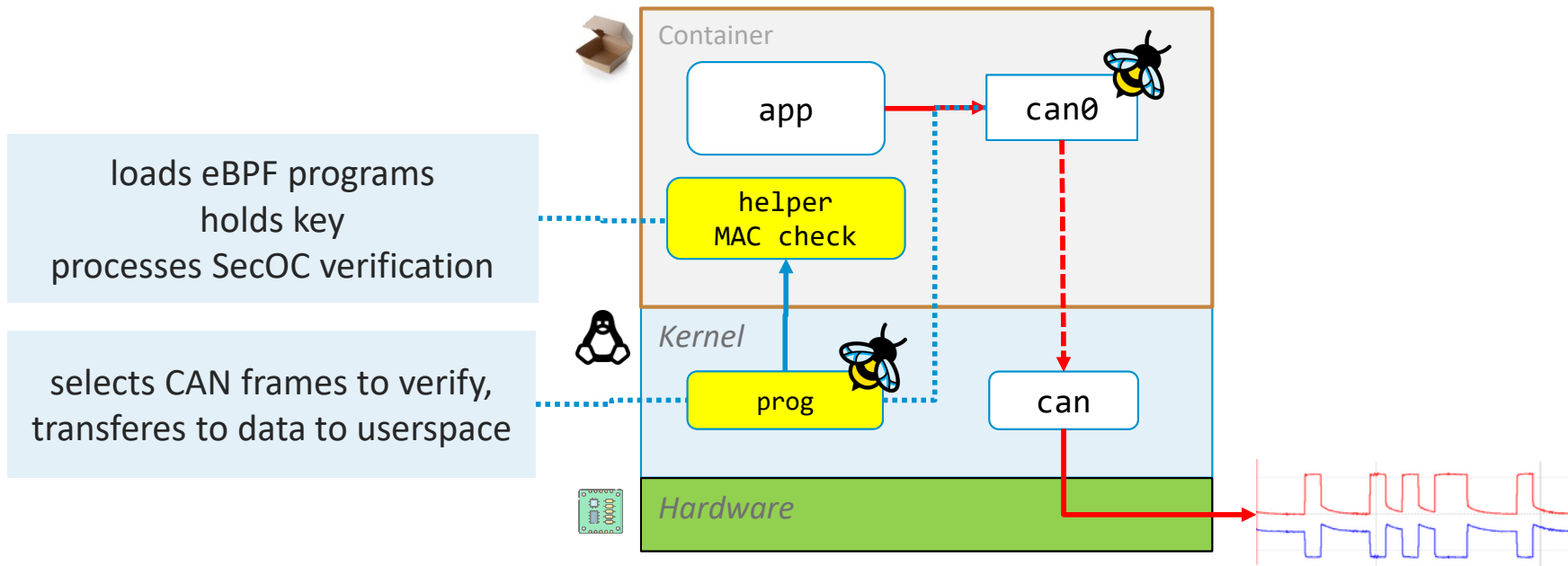
for Cryptographic verification of CAN traffic



SecOC: Integrity and Authenticity on CAN



Experiment 5: User space helper



Influence of an eBPF Program w/ userspace helper

SecOC Helper (read-only), no re-inject



- read-only, we can't wait for the user-space to terminate.
- the CAN frame isn't dropped and re-injected.

7 layer OSI model | Unified Diagnostic Services (UDS)

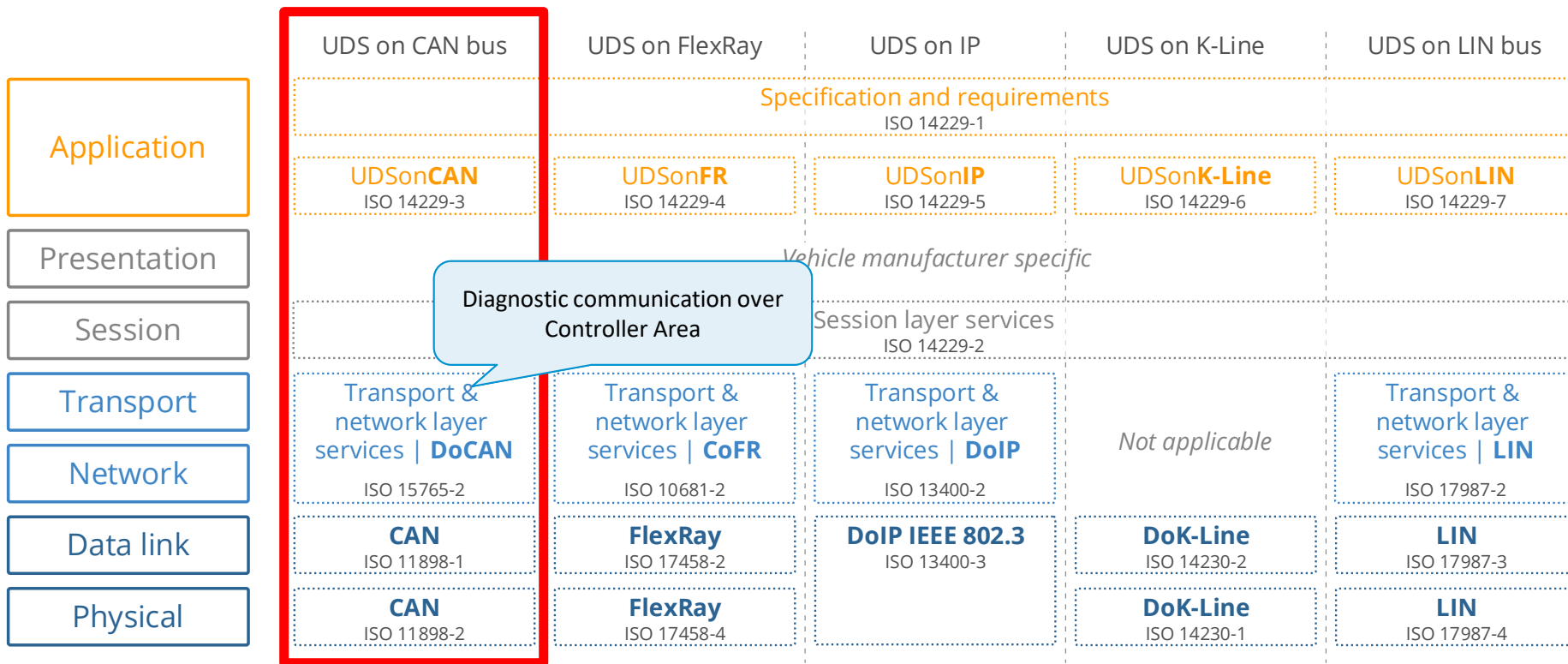
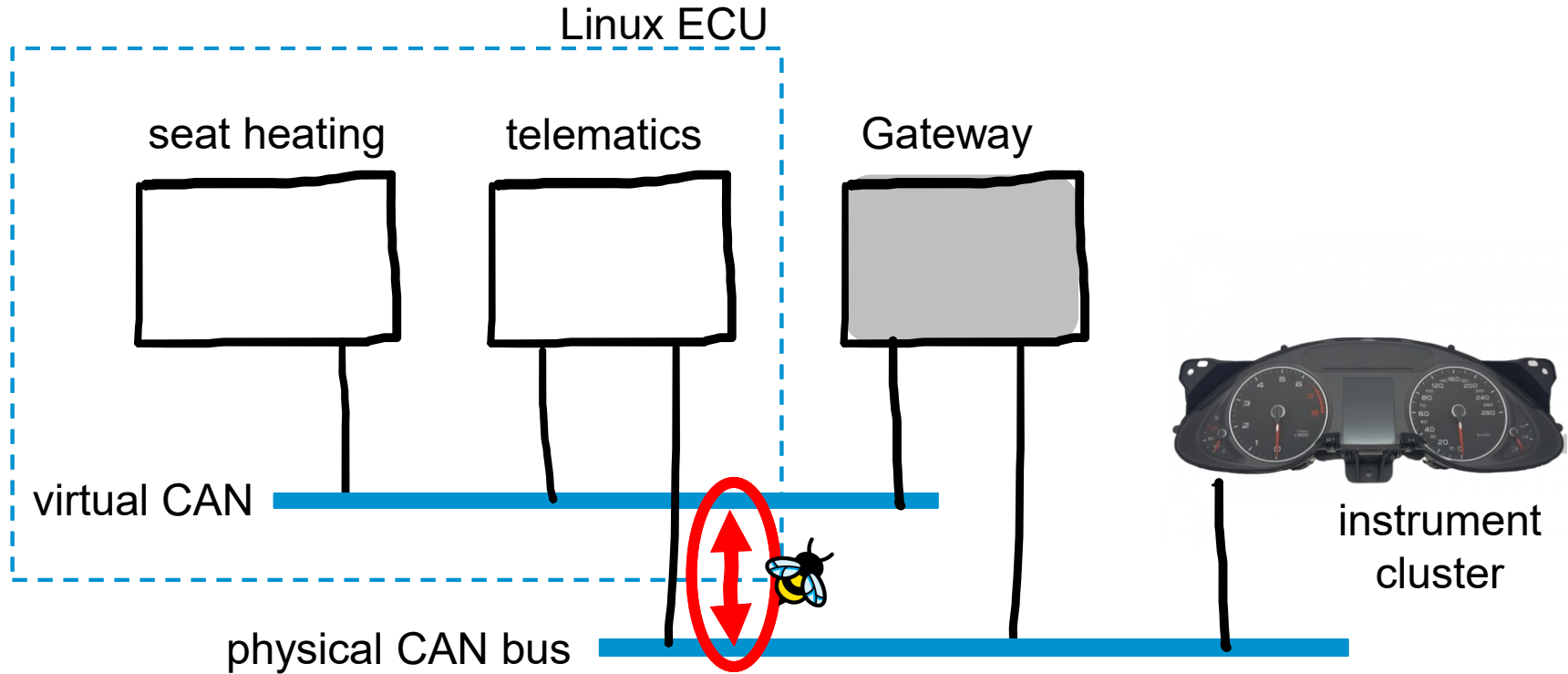


Image: <https://www.csselectronics.com/pages/uds-protocol-tutorial-unified-diagnostic-services>

Routing and Forwarding CAN Frames

Routing of CAN Traffic using eBPF



Routing and Forwarding of CAN frames

CANGW(1)

User Commands

NAME

cangw - manual page for cangw 2023.03-1

SYNOPSIS

cangw [*options*]

DESCRIPTION

cangw - manage PF_CAN netlink gateway.

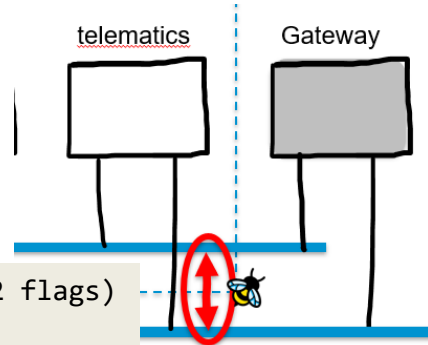
Commands:

- A** (add a new rule)
- D** (delete a rule)

- the can-gw Kernel module is existing infrastructure to route CAN traffic between interfaces
- the cangw user space tool configures iptables-like rules to forward/filter CAN frames to an interface
- it supports a set of bitmasked-based functions for filtering
- limitations on bitmasks to create more sophisticated filters

Can we do **bpf_clone_redirect()** to copy the skb to another CAN interface?

eBPF-CANGW: the packets drop



```
static int __bpf_redirect_common(struct sk_buff *skb, struct net_device *dev, u32 flags)
{
    /* Verify that a link layer header is carried */
    if (unlikely(skb->mac_header >= skb->network_header || skb->len == 0)) {
        kfree_skb(skb);
        return -ERANGE;
    }

    bpf_push_mac_rcsum(skb);
    return flags & BPF_F_INGRESS ?
        __bpf_rx_skb(dev, skb) : __bpf_tx_skb(dev, skb);
}

static int __bpf_redirect(struct sk_buff *skb, struct net_device *dev, u32 flags)
{
    if (dev_is_mac_header_xmit(dev))
        return __bpf_redirect_common(skb, dev, flags);
    else
        return __bpf_redirect_no_mac(skb, dev, flags);
}
```

```

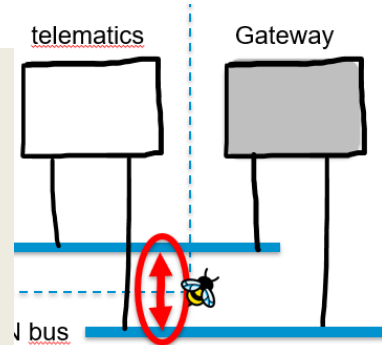
static int __bpf_redirect_no_mac(struct sk_buff *skb, struct net_device *dev, u32 flags)
{
    unsigned int mlen = skb_network_offset(skb);

    if (unlikely(skb->len <= mlen)) {
        kfree_skb(skb);
        return -ERANGE;
    }

    if (mlen) {
        __skb_pull(skb, mlen);

        /* At ingress, the mac header has already been pulled once.
         * At egress, skb_postpull_rcsum has to be done in case that
         * the skb is originated from ingress (i.e. a forwarded skb)
         * to ensure that rcsum starts at net header.
         */
        if (!skb_at_tc_ingress(skb))
            skb_postpull_rcsum(skb, skb_mac_header(skb), mlen);
    }
    skb_pop_mac_header(skb);
    skb_reset_mac_len(skb);
    return flags & BPF_F_INGRESS ?
        __bpf_rx_skb_no_mac(dev, skb) : __bpf_tx_skb(dev, skb);
}

```



bpf_redirect(): Gateway in eBPF

```
static int __bpf_redirect(struct sk_buff *skb, struct net_device *dev,  
                        u32 flags)
```

```
{  
    if (dev_is_mac_header_xmit(dev))  
        return __bpf_redirect_common(skb, dev, flags);  
    else  
        return __bpf_redirect_no_mac(skb, dev, flags);  
}
```

```
static inline bool dev_is_mac_header_xmit(const struct net_device *dev)  
{
```

```
    switch (dev->type) {  
        case ARPHRD_TUNNEL:  
        case ARPHRD_TUNNEL6:  
        case ARPHRD_SIT:  
        case ARPHRD_IPGRE:  
        case ARPHRD_IP6GRE:  
        case ARPHRD_VOID:  
        case ARPHRD_NONE:  
        case ARPHRD_RAWIP:  
        case ARPHRD_PIMREG:
```

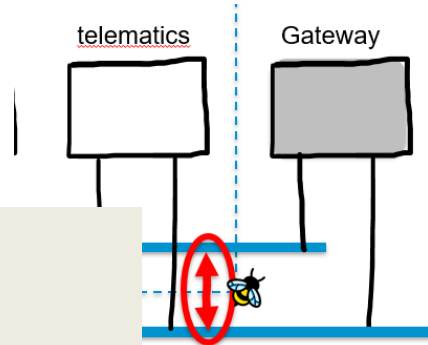
```
        /* PPP adds its l2 header automatically in ppp_start_xmit().
```

```
         * This makes it look like an l3 device to __bpf_redirect() and tcf_mirred_init().
```

```
         */
```

```
        case ARPHRD_PPP:
```

[net/core/filter.c](#)
[include/linux/if_arp.h](#)



Discussion and RFC

- Should be CAN be supported in bpf_redirect?
What part needs to be extended?
 - dev_is_mac_header_xmit
 - __bpf_redirect
- can-gw works, do we need another way in eBPF?

Serial Peripheral Interface



Example Security Use Case: SPI

- Secure an update process
- Check unusual write operations to the flash device

Method

- Prevent a process from opening a SPI device
- Observe write operations on an SPI device
- Check writes with signatures (hashes, signature, CRC)

Example: OpenWRT

```
# /lib/upgrade/common.sh, simplified

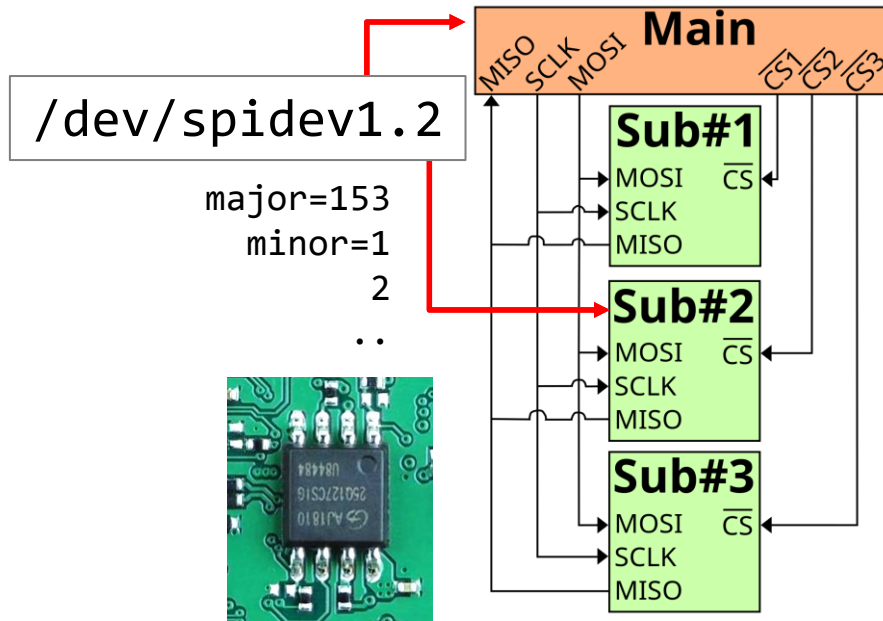
default_do_upgrade()
{
    # write image and append saved config as a jffs2 blob
    sync
    if [ -n "$UPGRADE_BACKUP" ]; then
        get_image "$1" | mtd -j "$UPGRADE_BACKUP" write - "${PART_NAME:-firmware}"
    else
        get_image "$1" | mtd write - "${PART_NAME:-firmware}"
    fi
}
```

lib/upgrade/common.sh
lib/upgrade/fwtool.sh

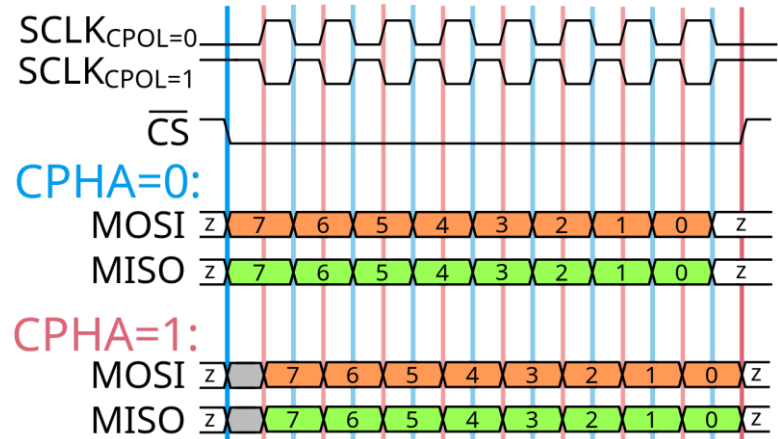
Serial Peripheral Interface

- SPI core drivers/spi/spi.c (struct spi_controller)
 - hardware spi-bcm2835.c, spi-imx.c
 - spi_message, spi_transfer, spi_write/spi_read
- Protocol drivers
 - mtd (spi-nor, spinand): /dev/mtd0, /dev/mtdblock1
 - mmc (SD): /dev/mmcblk0, /dev/mmcblk0p1
- Userspace interface
 - Spidev /dev/spidev1.2

SPI topology and signals



`spi->controller->bus_num`



Images: https://en.wikipedia.org/wiki/Serial_Peripheral_Interface

spi (Linux Kernel): <https://www.kernel.org/doc/Documentation/spi/spi-summary>

Spidev: <https://www.kernel.org/doc/Documentation/spi/spidev>

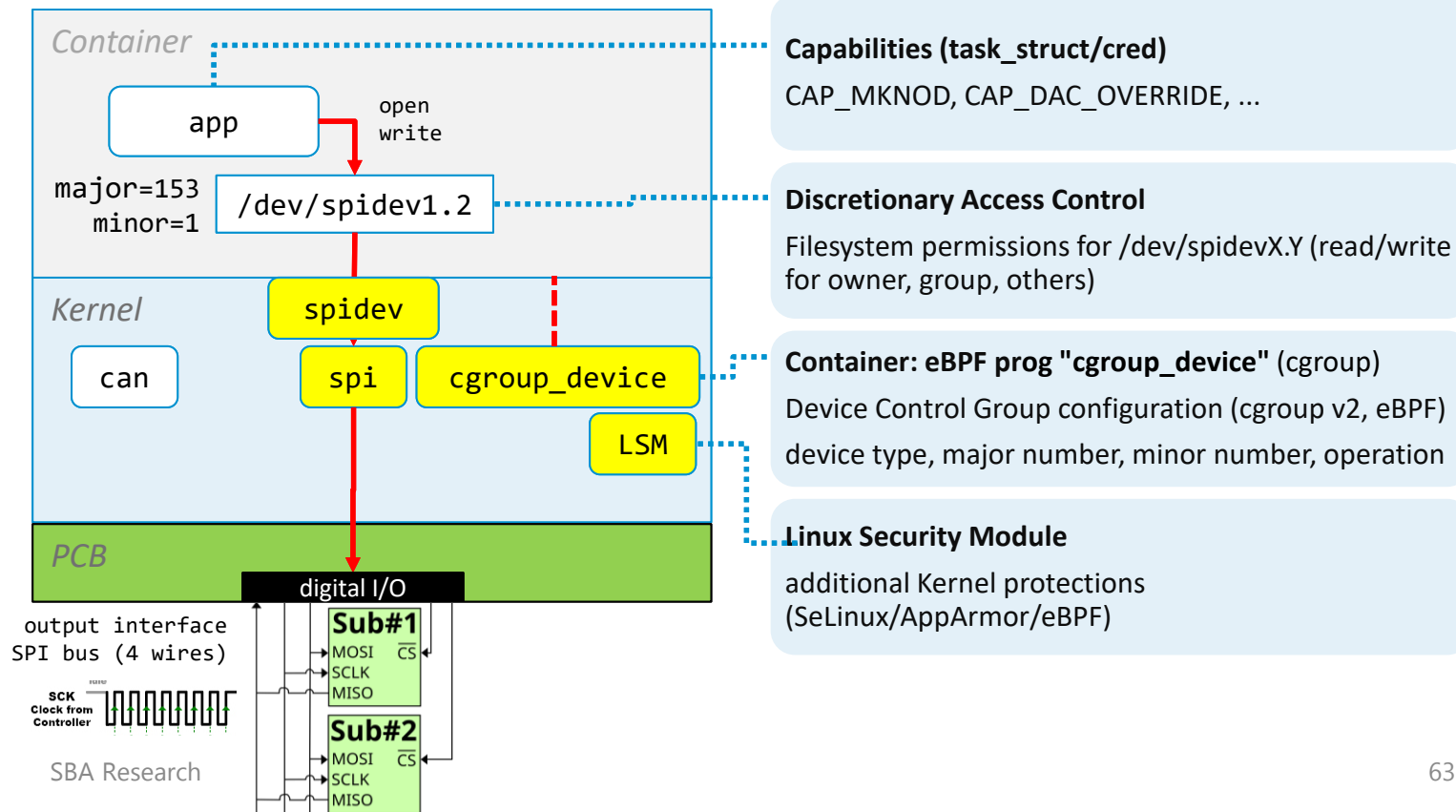
Python Library (py-spidev): <https://pypi.org/project/spidev/>

Spidev

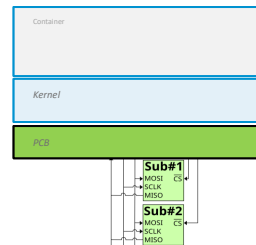
- device node `/dev/spidevB.C`
 - B is the controller
 - C is the chipselect
- major number of spidev is 153 with a dynamically chosen minor device number
- dev nodes are populated by udev/container runtime
- protected by CAP/DAC/eBPF/MAC
- spidev resolves actual SPI device

<https://docs.kernel.org/spi/spidev.html>

Security Layers for SPI bus access (Linux)



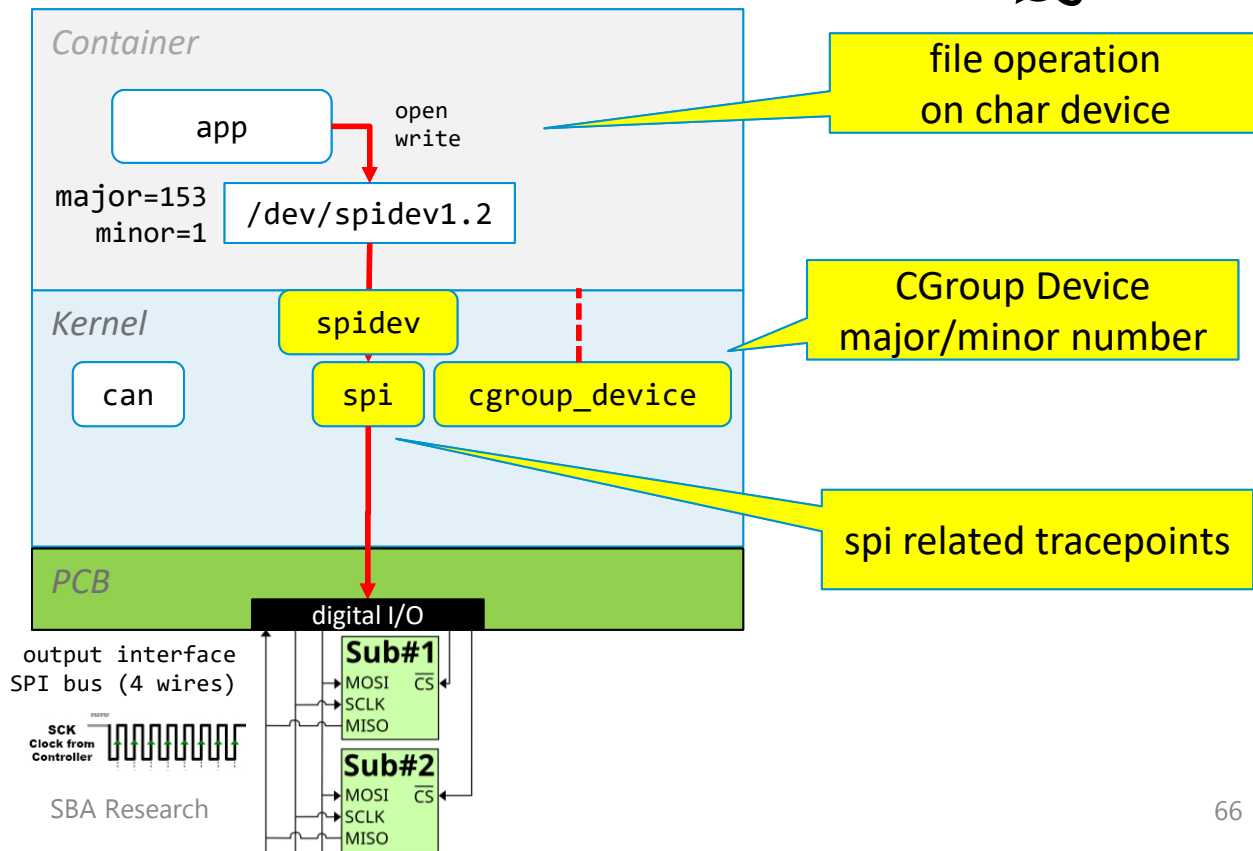
Discretionary Access Control in Practice



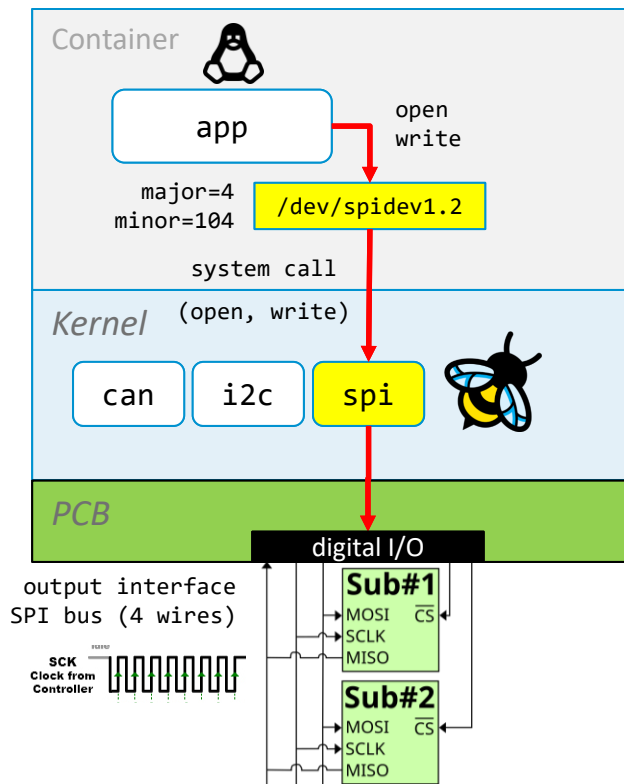
```

crw-rw-rw- 1 root tty      5,  2 Jun  4 15:20 ptmx
drwxr-xr-x 2 root root          0 Jan  1 1970 pts
crw-rw-rw- 1 root root      1,  8 May 25 08:17 random
crw-rw-r-- 1 root netdev 10, 242 May 25 08:17 rfskill
drwxrwxrwt 2 root root          40 Jan 26 21:48 shm
drwxr-xr-x 3 root root        220 Jun  4 14:51 snd
crw-rw---- 1 root spi      153,  0 Jun  4 14:51 spidev0.0
crw-rw---- 1 root spi      153,  1 Jun  4 14:51 spidev0.1
lrwxrwxrwx 1 root root          15 Jan  1 1970 stderr → /proc/self/fd/2
lrwxrwxrwx 1 root root          15 Jan  1 1970 stdin  → /proc/self/fd/0
lrwxrwxrwx 1 root root          15 Jan  1 1970 stdout → /proc/self/fd/1
crw-rw-rw- 1 root tty      5,  0 Jun  4 15:08 tty
crw--w---- 1 root tty      4,  0 May 25 08:17 tty0
crw--w---- 1 root tty      4,  1 Jun  4 14:53 tty1
    
```

eBPF for SPI bus access



eBPF for SPI bus access (Linux)



SBA Research

```
SEC("tracepoint/spi/spi_transfer_start")
int transfer_start(
    struct spi_transfer_start_args *ctx)
{
    // ...
    bpf_probe_read(&tx_buffer,
        tx_length,
        (void *)((char *)ctx + tx_offset));
    check_signature(tx_buffer);
    return 0;
}

SEC("cgroup/dev")
int spifilter(struct bpf_cgroup_dev_ctx *ctx)
{
    // ...
    if(ctx->major == 153 && // spi
        ctx->minor == 1 && // bus number
        (access & BPF_DEVC_G_ACC_WRITE))
        return 0; // block write

    return 1; // allow
}
```

Commands for a Flash

Table 7.1. Commands (Standard/Dual/Quad SPI)

Command Name	Byte 1	Byte 2	Byte 3	Byte 4	Byte 5	Byte 6	n-Bytes
Write Enable	06H						
Write Disable	04H						
Volatile SR Write Enable	50H						
Read Status Register-1	05H	(S7-S0)					(continuous)
Read Status Register-2	35H	(S15-S8)					(continuous)
Read Status Register-3	15H	(S23-S16)					
Write Status Register-1	01H	(S7-S0)					
Write Status Register-2	31H	(S15-S8)					
Write Status Register-3	11H	(S23-S16)					
Read Data	03H	A23-A16	A15-A8	A7-A0	(D7-D0)	(Next byte)	(continuous)
Fast Read	0BH	A23-A16	A15-A8	A7-A0	dummy	(D7-D0)	(continuous)
Dual Output Fast Read	3BH	A23-A16	A15-A8	A7-A0	dummy	(D7-D0) ⁽¹⁾	(continuous)
Dual I/O Fast Read	BBH	A23-A8 ⁽²⁾	A7-A0 M7-M0 ⁽²⁾	(D7-D0) ⁽¹⁾			(continuous)

Datasheet of a Dual and Quad Serial Flash GD25Q127C

Memory Layout of the Target

TP-Link WR1043ND Flash Layout					
Layer0	raw NOR flash memory chip (m25p80 W spi0.0 : m25p64) 8192 KiB				
Layer1	mtd0 <i>u-boot</i> 128 KiB	mtd5 <i>firmware</i> 8000 KiB			mtd4 <i>art</i> 64 KiB
Layer2		mtd1 <i>kernel</i> 1280 KiB	mtd2 <i>rootfs</i> 6720 KiB		
mountpoint			/		
filesystem			OverlayFS		
Layer3				mtd3 <i>rootfs_data</i> 5184 KiB	
Size in KiB	128 KiB	1280 KiB	1536 KiB	5184 KiB	64 KiB
Name	<i>u-boot</i>	<i>kernel</i>		<i>rootfs_data</i>	<i>art</i>
mountpoint	<i>none</i>	<i>none</i>	/rom	/overlay	<i>none</i>
filesystem	<i>none</i>	<i>none</i>	SquashFS	JFFS2	<i>none</i>

TP-Link WR1043ND Flash Layout, OpenWrt, <https://openwrt.org/docs/techref/flash.layout>

```

#include "vmlinux.h"
#include <bpf/bpf_helpers.h>

#define DEBUG 1

enum {
    BPF_DEVCG_ACC_MKNOD    = (1ULL << 0),
    BPF_DEVCG_ACC_READ     = (1ULL << 1),
    BPF_DEVCG_ACC_WRITE    = (1ULL << 2),
};

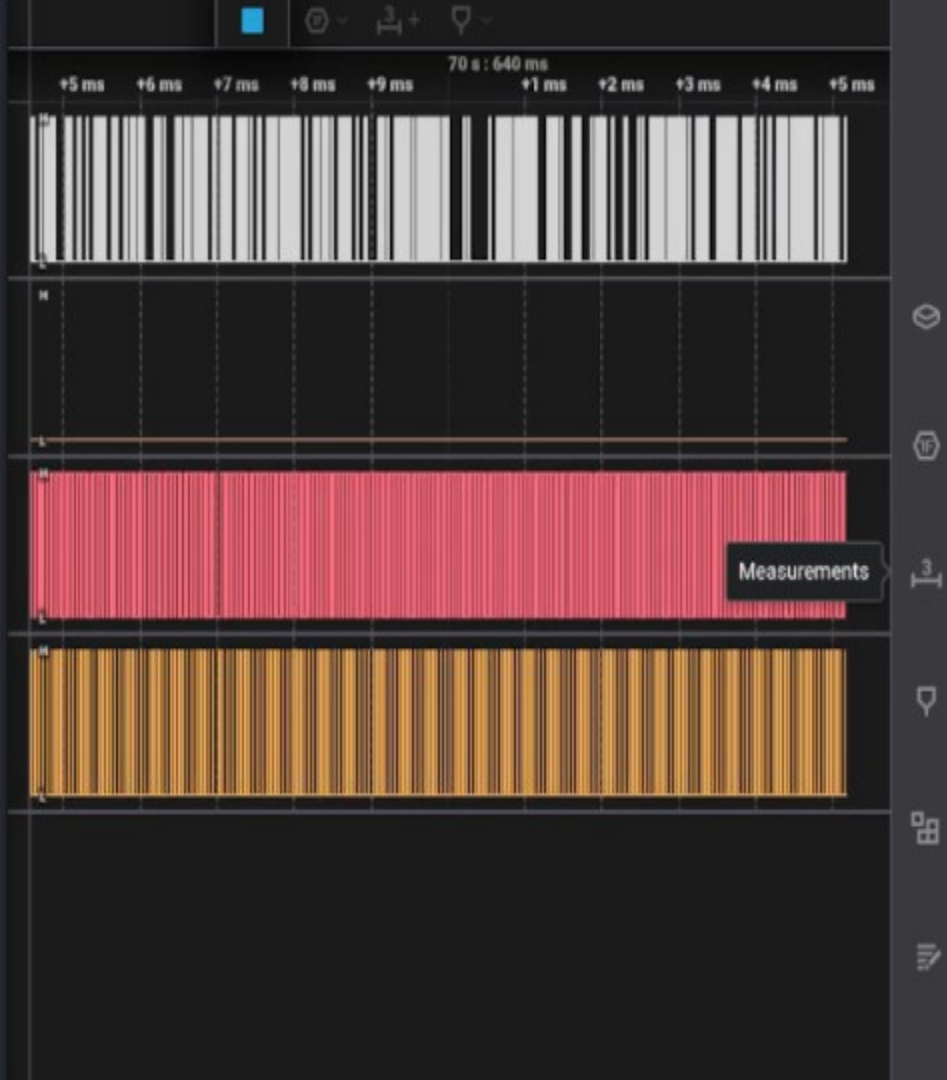
enum {
    BPF_DEVCG_DEV_BLOCK    = (1ULL << 0),
    BPF_DEVCG_DEV_CHAR     = (1ULL << 1),
};

SEC("cgroup/dev")
int spifilter(struct bpf_cgroup_dev_ctx *ctx)
{
    short type = ctx->access_type & 0xFFFF;
    short access = ctx->access_type >> 16;

#ifdef DEBUG
    char fmt[] = " %d:%d  \n";

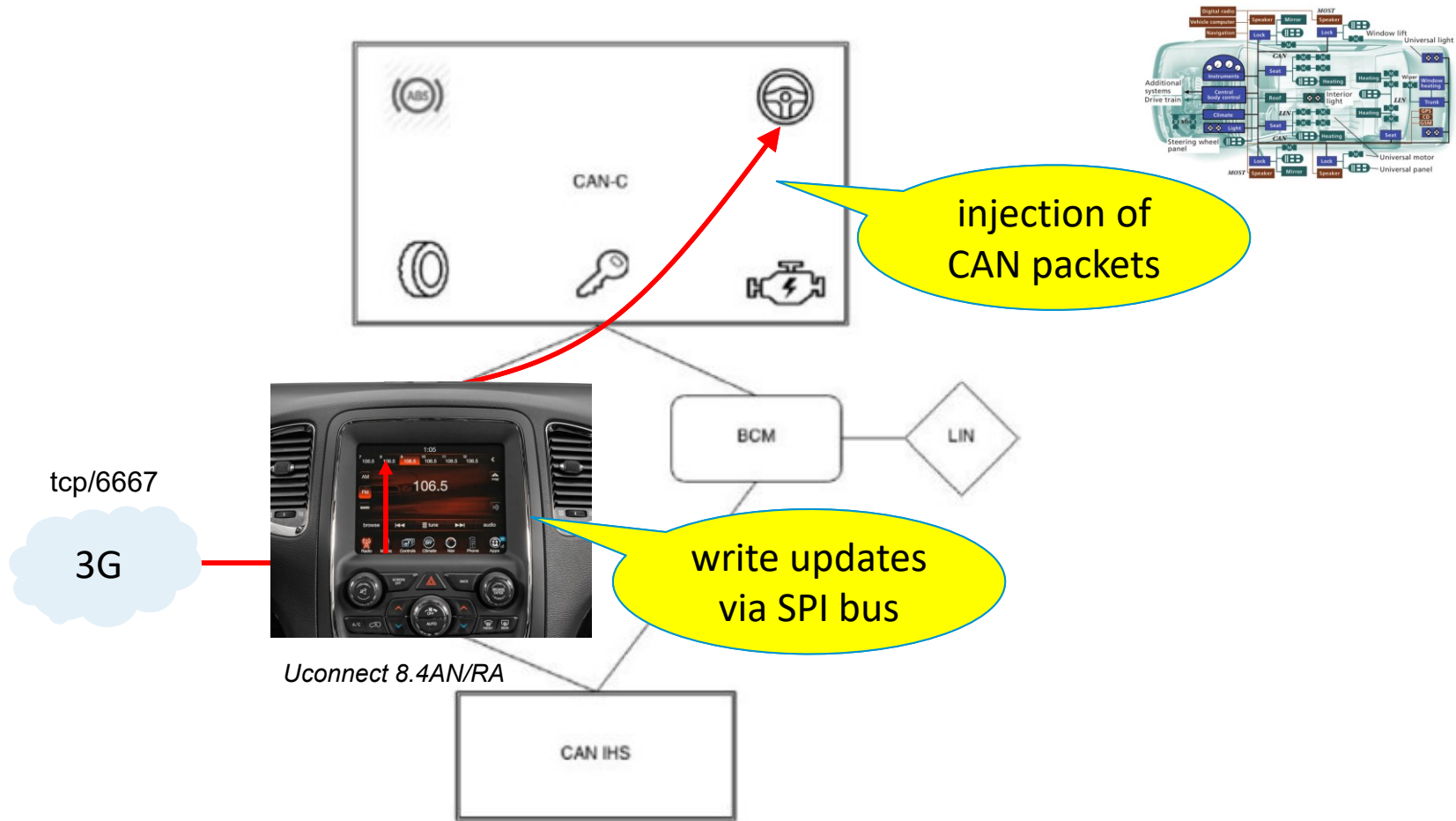
    switch (type) {
    case BPF_DEVCG_DEV_BLOCK:

```



- trigger
 - flash addresses/offsets
 - write requests on the nand flash
- check for
 - magic bytes in images
- we could identify write requests containing firmware
- create a
- changed magic or arbitrary write would defeat detection

Conclusion

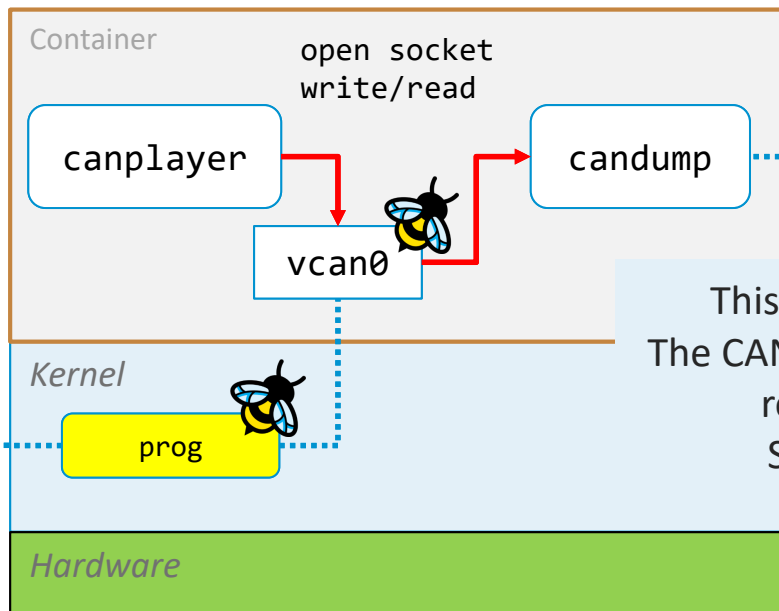


Experiment 1: XDP filter for CAN CC (2.0)

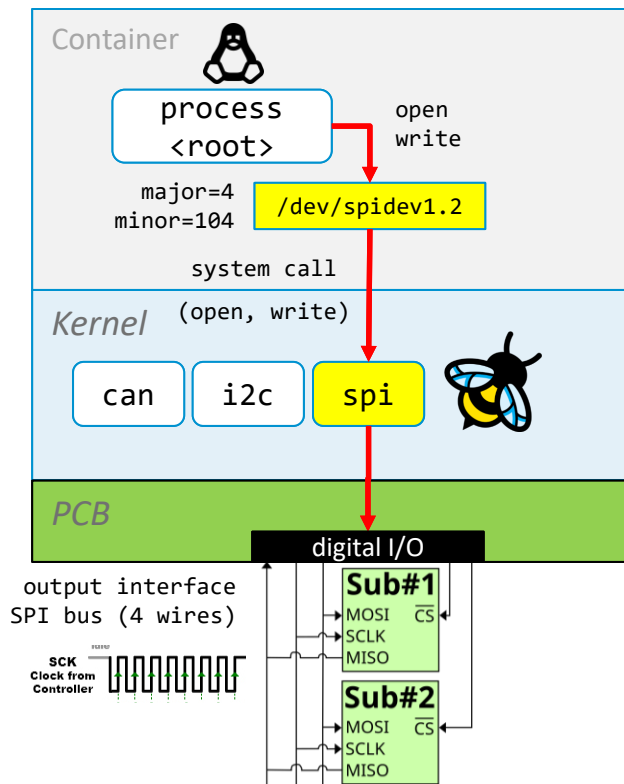
Use CAN traffic from a capture-file and send it into a virtual interface. Receive the echoed CAN frames with candump.

uses a capture file of a real car and replays it using the existing timestamps

dissects the CAN protocol and filters



eBPF for SPI bus access (Linux)



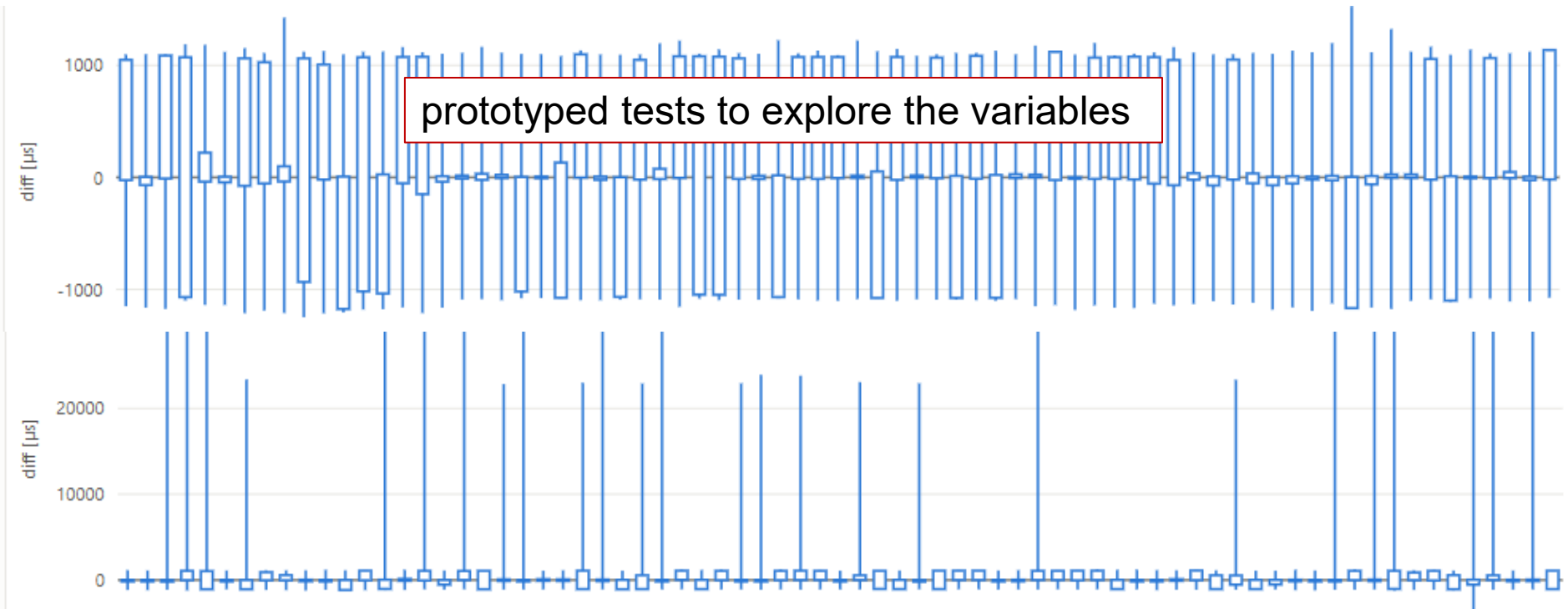
SBA Research

```
SEC("tracepoint/spi/spi_transfer_start")
int transfer_start(
    struct spi_transfer_start_args *ctx)
{
    // ...
    bpf_probe_read(&tx_buffer,
        tx_length,
        (void *)((char *)ctx + tx_offset));
    check_signature(tx_buffer);
    return 0;
}

SEC("cgroup/dev")
int spifilter(struct bpf_cgroup_dev_ctx *ctx)
{
    // ...
    if(ctx->major == 153 && // spi
        ctx->minor == 1 && // bus number
        (access & BPF_DEVCG_ACC_WRITE))
        return 0; // block write

    return 1; // allow
}
```

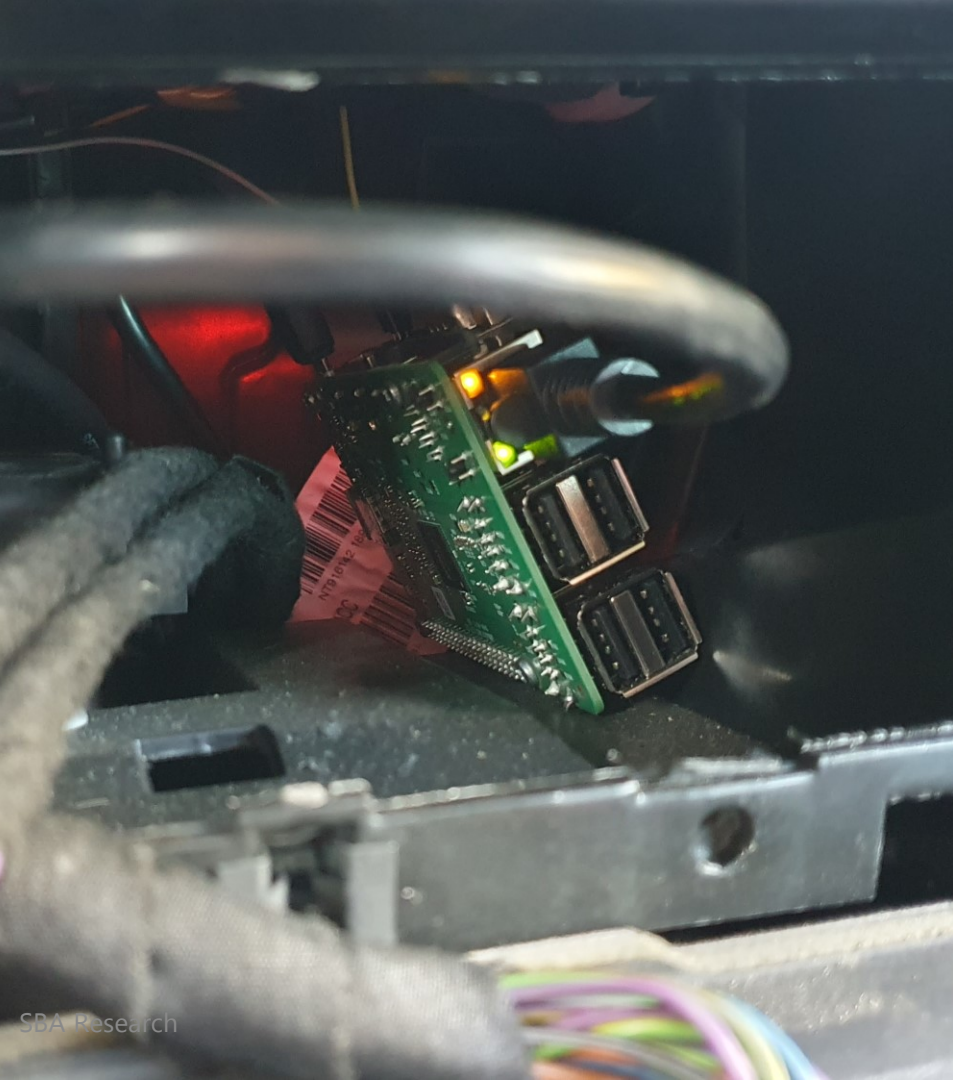
Influence of an extended eBPF Program on CAN



note, that the static diagrams were produced with insufficient data.

Open Questions

- What exactly is influencing the performance?
 - What coding constructs introduce the delays?
 - XDP vs. TC?
- Performance impact of native code vs. JIT-ed?
- Should be CAN be supported in bpf_redirect?
What part needs to be extended?



Reinhard Kugler

SBA Research

rkugler@sba-research.org



thanks to Dimitrij Klesev who supported this work.

