

CAN-Lab White Paper

Summary, motivation and test set-up

blog.andrelademann.de

Two ESP32-C3 boards with SN65HVD230 transceivers carry a steady **4,303 CAN frames per second** at 500 kbit/s, each with 8 data bytes. During the measurement there were **no counter gaps, no receive-queue overruns and no bus errors**. That is about 116 bit times per frame: the bus was effectively saturated, so the wire set the limit, not the hardware.

MOTIVATION

I am rebuilding the electrics of my Peugeot 304 S: distributed ESP32 nodes with sensors, talking over CAN, and a Raspberry Pi 5 as the central computer. Before any of that goes into the car, I wanted to know how the bus behaves on cheap maker hardware, under load rather than just idling.

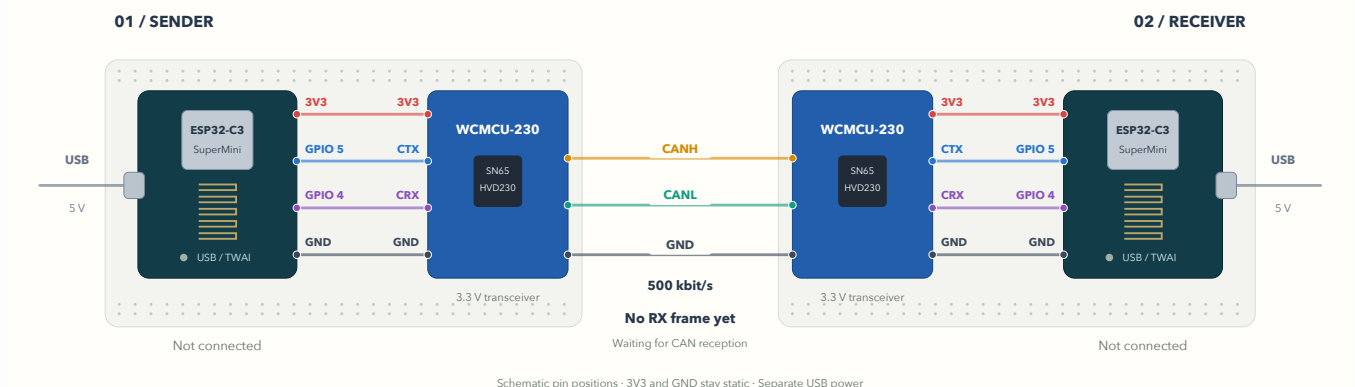
That is why I built [CAN-Lab](#): a Web Serial console in the browser that reads two boards at once, controls the sender and shows throughput and error counters live. The code is open source under GPL-3.0 on [GitHub](#).

PARTS

Component	Qty	Role
ESP32-C3 SuperMini	2	Sender and receiver, built-in CAN controller (TWAI)
SN65HVD230 transceiver module, 3.3 V	2	Converts logic levels to CAN_H/CAN_L
Breadboard and jumper wires	1 set	Wiring
USB cables (data, not charge-only)	2	Power and Web Serial

The exact boards and modules I used are linked in the [blog post](#). An ESP8266 has no CAN controller and will not work here.

HOW THE TWO NODES TALK



WIRING, IDENTICAL ON BOTH SIDES

ESP32-C3	Transceiver	Between the modules
3V3	3V3	CANH to CANH
GND	GND	CANL to CANL
GPIO 5	CTX	GND to GND
GPIO 4	CRX	Each board on its own USB supply

Bus parameters: 500 kbit/s, standard ID 0x123 (11 bits), 8 data bytes per frame (4-byte counter, 4-byte test ID, little endian). Transmit queue 32 frames, receive queue 128 frames.

CAN-Lab White Paper

Breadboard layout and methodology

blog.andrelademann.de

BREADBOARD LAYOUT

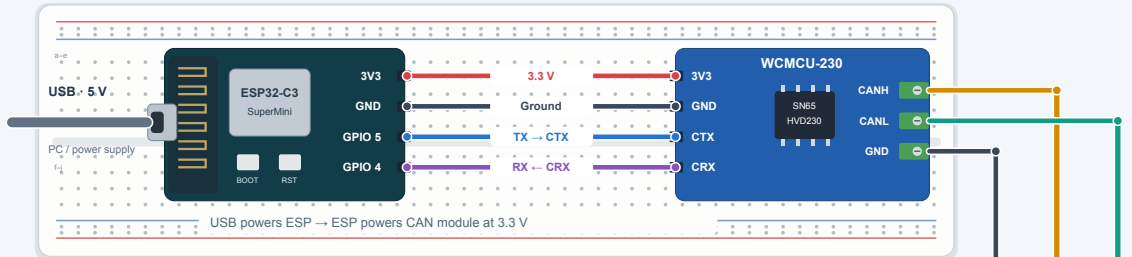
Two boards. One CAN bus.

ESP32-C3 SuperMini + WCMCU-230 / SN65HVD230 · 500 kbit/s

Wiring diagram: components and pin positions are schematic. Follow the pin labels on the actual board.

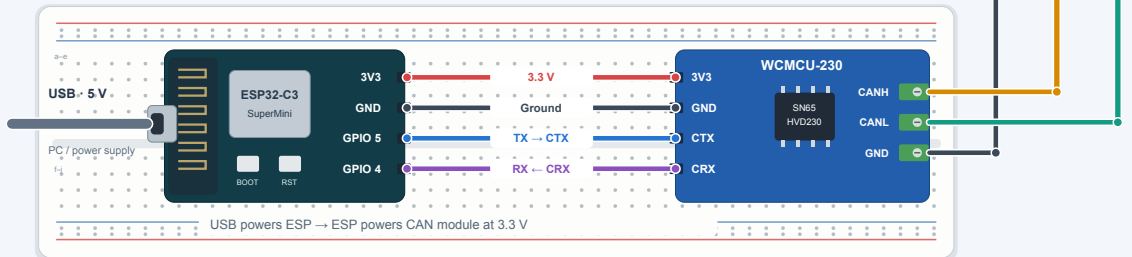
01 SENDER

sender.ino · sends counter with ID 0x123



02 RECEIVER

receiver.ino · receives frames and blinks



How to connect

CANH ↔ CANH · CANL ↔ CANL · GND ↔ GND

Twist CANH and CANL for longer connections.

Power both ESPs separately via USB; no 5V/3V3 jumper.

Termination & checks

WCMCU-230 modules have built-in bus termination.

No additional external resistors are needed.

With power off, measure CANH to CANL: total ≈ 60 Ω.

Source: repository README / GPIO configuration · SN65HVD230: TI datasheet · Power CAN modules at 3.3 V only.

Also available as an [interactive breadboard on Wokwi](#) – a wiring aid, not an electrical CAN simulation.

METHODOLOGY

- 1. Handshake.** Both boards report role, protocol version and bitrate (LAB HELLO). If the bitrates differ, no test starts.
- 2. Arming.** The receiver is prepared first; the sender only starts once the receiver confirms it is ready.
- 3. Generating load.** The sender tries to queue the target rate. Whatever the transmit queue does not accept counts as rejected, without incrementing the payload counter, so rejected attempts never create artificial gaps.
- 4. Measuring.** Both boards report their counters about once per second from their own firmware measurement window, not from USB log arrival times.
- 5. Detecting gaps.** The receiver checks the sequence counter. A missing value counts as a gap.

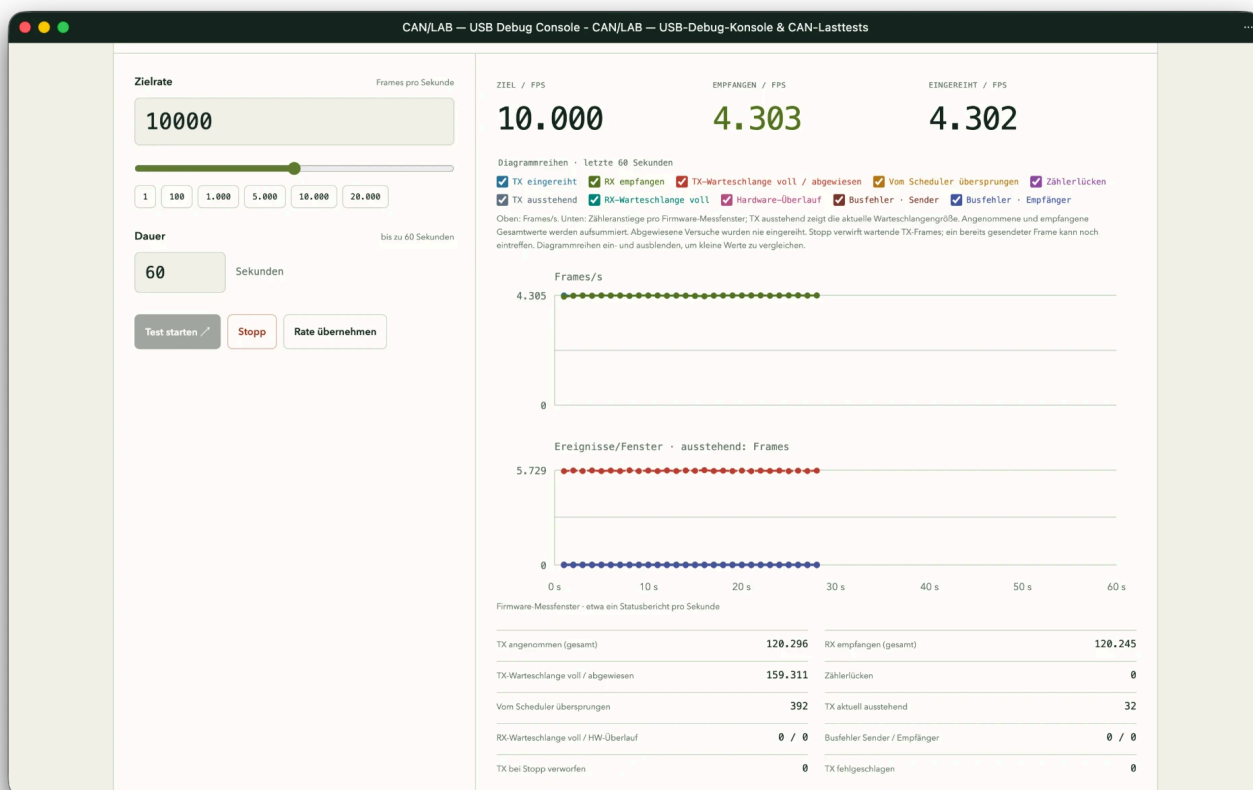
The target rate is offered load and may exceed the bus capacity. That is exactly the point of this test.

CAN-Lab White Paper

Results

blog.andrelademann.de

LOAD TEST: 10,000 FRAMES/S OFFERED, 60 S PLANNED



Captured while the test was running, about 28 seconds in. The console labels are German; the table translates them.

Measurement	Value	Measurement	Value
Target rate	10,000 frames/s	Counter gaps	0
Received	4,303 frames/s	RX queue full / HW overrun	0 / 0
Queued	4,302 frames/s	Bus errors sender / receiver	0 / 0
TX accepted (total)	120,296	TX failed	0
RX received (total)	120,245	TX currently pending	32
TX queue full / rejected	159,311	Skipped by scheduler	392

The received-frames line stays flat at roughly 4,300 frames/s for the whole run, with no dips and no decline over time.

CAN-Lab White Paper

Interpretation and limits

blog.andrelademann.de

THE BUS WAS FULL

A classic CAN frame with an 11-bit ID and 8 data bytes takes 108 bits, or 111 bits including the interframe space. Bit stuffing can add up to 24 more bits. That gives a theoretical ceiling of:

Bitrate	No stuffing	Worst-case stuffing
500 kbit/s	approx. 4,505 frames/s	approx. 3,704 frames/s
1 Mbit/s	approx. 9,009 frames/s	approx. 7,407 frames/s

500,000 bit/s divided by 4,303 frames/s gives about 116 bit times per frame: 111 bits plus an average of roughly 5 stuff bits. The measured value sits squarely in the expected range, and the bus had practically no idle bit time left.

WHAT THE COUNTERS SHOW

The sender was not the bottleneck. Accepted plus rejected attempts add up to 279,607 – almost exactly 28 seconds at 10,000 attempts per second. The queue filled up because the bus could not take frames any faster.

No data loss. Zero counter gaps: every queued frame arrived, in order. The difference of 51 frames between the totals is plausibly explained by the 32 frames still pending and the unsynchronised measurement windows of the two boards.

LIMITS OF THIS TEST

Two nodes, one ID. Arbitration between several senders was not tested.

Short wires on a breadboard. Cable length, stubs and in-vehicle interference play no part here.

A snapshot. One run, captured about 28 of 60 seconds in. Solid conclusions need repeated and longer runs.

Full load is not an operating target. Real vehicle networks are planned with plenty of headroom, so that high-priority messages always get through.

Missing final frames go unnoticed. If the last frames of a test are lost, the counter alone cannot detect it.

CAN-Lab White Paper

Build guide, part 1: hardware and firmware

blog.andrelademann.de

1. GET THE HARDWARE

Two ESP32-C3 boards, two SN65HVD230 modules, a breadboard, jumper wires and two USB data cables (see the parts list). Make sure the board carries an **ESP32-C3**: the sketches are written for the C3 SuperMini. Other ESP32 variants with TWAI work in principle but need adjusted pins.

2. WIRE IT UP

Solder the pin headers, place both boards and modules on the breadboard and wire them as shown in the wiring table. Then, with the power off, measure the resistance between CANH and CANL: **about 60 Ω** means both 120 Ω terminators on the modules are in place. About 120 Ω means one end is missing its termination.

Never connect 3V3 to 3V3 between the boards, and never power the transceivers at 5 V.

3. FLASH THE FIRMWARE

```
git clone https://github.com/vergissberlin/example-canbus.git
cd example-canbus

# Flash the sender (first board connected)
pio run -e sender -t upload

# Flash the receiver (second board connected)
pio run -c receiver.ini -e receiver -t upload
```

Requires [PlatformIO](#). With both boards connected at once, pick the port with `--upload-port`.

THE CORE OF THE SENDER CONFIGURATION

```
#include "driver/twai.h"

constexpr gpio_num_t CAN_TX_PIN = GPIO_NUM_5;
constexpr gpio_num_t CAN_RX_PIN = GPIO_NUM_4;

void setup() {
    twai_general_config_t g = TWAI_GENERAL_CONFIG_DEFAULT(
        CAN_TX_PIN, CAN_RX_PIN, TWAI_MODE_NORMAL);
    g.tx_queue_len = 32;
    twai_timing_config_t t = TWAI_TIMING_CONFIG_500KBITS();
    twai_filter_config_t f = TWAI_FILTER_CONFIG_ACCEPT_ALL();
    twai_driver_install(&g, &t, &f);
    twai_start();
}
```

Pins, transmit queue, 500 kbit/s and a filter that accepts every ID.

CAN-Lab White Paper

Build guide, part 2: CAN-Lab and the load test

blog.andrelademann.de

4. CONNECT CAN-LAB

1. Open canlab.andrelademann.de in **Chrome or Edge**. Safari does not support Web Serial.
2. Plug in the receiver first and click **Connect USB**, then do the same for the sender.
3. No output? Briefly press **RESET** (not BOOT) on that board.

USB data stays in your browser. CAN-Lab can be installed as an app and then also works offline.

5. CHECK THE CONNECTION

In normal operation the sender transmits one frame per second. If the **RX ID 0x123** counter on the receiver goes up, CAN communication works. A **TX ... ok** on the sender only confirms queueing, not reception.

6. RUN THE LOAD TEST

1. Start gently: **100 frames/s for 10 seconds**.
2. If that runs without gaps or errors, step up to 1,000, then 5,000 and finally 10,000 frames/s.
3. Set the duration to 60 seconds and watch where the received rate levels off. Around 4,300 frames/s at 500 kbit/s without gaps means your set-up runs at the limit of the bus.

7. TROUBLESHOOTING

No frames at the receiver: is the shared ground missing? That is the most common cause.

Bus errors climb immediately: CANH and CANL swapped, or termination missing – measure the resistance.

No serial port: a charge-only cable, or another serial monitor is holding the port.

Gaps at high rates: missing USB log lines can fake gaps. The firmware counters are what counts.

WHAT COMES NEXT

Next I will test 1 Mbit/s, several senders with different IDs and longer twisted-pair cables. Then the set-up moves to the Raspberry Pi 5 with a CAN HAT and SocketCAN, and into the car. If you rebuild the test, send me your numbers.

SOURCES

CAN-Lab

<https://canlab.andrelademann.de/>

vergissberlin/example-canbus – sketches and CAN-Lab source, GPL-3.0

<https://github.com/vergissberlin/example-canbus>

History of CAN technology – CAN in Automation

<https://www.can-cia.org/can-knowledge/history-of-can-technology>

CAN bus – Wikipedia, frame format and bit stuffing

https://en.wikipedia.org/wiki/CAN_bus