# Neue CAN Konfiguration

Dieses Dokument zeigt die Netzwerk-Änderungen in Debian Bookworm / Trixie (also auch dem Raspberry Pi OS) im Bezug auf CAN. Es behandelt die Umstellung von **ifupdown** zu **systemd-networkd**, und zeigt wie man Probleme analysiert, Konfigurationen erstellt und Konflikte löst.

## YouTube Video #141


## Warum anders ab Bookworm

Seit Bookworm ist **ifupdown** nicht mehr standardmäßig installiert und wurde durch **systemd-networkd** ersetzt. Dies betrifft besonders CAN-Interfaces (Controller Area Network), die viel im 3D-Druck genutzt werden.

**Warum der Wechsel?** *systemd-networkd* ist modern, in systemd integriert, ermöglicht parallele Verarbeitung für schnellere Boot-Zeiten und reduziert Abhängigkeiten von alten Skripten.

**Vorteile**: Höhere Stabilität in serverartigen oder embedded Setups, bessere Fehlerbehandlung, Skalierbarkeit für komplexe Netzwerke wie Bridges oder VLANs.

**Nachteile**: Lernkurve für Legacy-Nutzer und mögliche Konflikte mit **NetworkManager**, das auf Raspberry Pi für WLAN/Ethernet läuft.

## Was ist das ... ?

- **ifupdown**: Ein klassisches, script-basiertes Tool aus den 1990er Jahren, das über Textdateien wie `/etc/network/interfaces` arbeitet. Es ist einfach für statische Konfigurationen (z. B. feste IPs, CAN-Queues), aber imperativ (Schritt-für-Schritt) und veraltet. In Bookworm nicht mehr Standard, muss manuell installiert werden (`apt install ifupdown`). Geeignet für Legacy, aber fehlt an Integration in systemd, was zu langsameren Boots und Fehlern führen kann Debian Network Config ifupdown Docs.
- **NetworkManager (NM)**: Ein dynamisches Tool für Desktops/Laptops, ideal für WiFi, VPNs, Hotplugging. Verwendet Verbindungen (via `nmcli` oder GUI wie `nm-applet`). Auf Raspberry Pi Bookworm Standard für WLAN/Ethernet, besonders mit GUI. Stark für dynamische Setups, aber

ressourcenintensiv und kann mit systemd-networkd kollidieren NetworkManager Docs Debian NM Wiki.

- **systemd-networkd**: Moderner, deklarativer Manager, integriert in systemd. Konfiguriert über `.network`- und `.link`-Dateien in `/etc/systemd/network/`. Leichtgewichtig, schnell, stabil für Server/IoT, Default in Bookworm für headless Setups. Ideal für CAN (Bitrate, Restarts direkt setzbar), aber ohne GUI Debian systemd-networkd systemd Docs.

**Vergleichstabelle**:

| Tool | Geschichte & Typ | Stärken | Schwächen | Raspberry Pi-Nutzung |
|---|---|---|---|---|
| ifupdown | 1990er, script-basiert | Einfach, statisch, niedrige Ressourcen | Veraltet, keine Parallelarbeit, imperativ | Optional für Legacy, manuell installieren |
| NetworkManager | Modern, dynamisch | GUI/TUI, Hotplugging, WiFi/VPN | Ressourcenintensiv, Konflikte möglich | Standard für Desktop/WLAN/Ethernet |
| systemd-networkd | Systemd-integriert, deklarativ | Schnell, stabil, skalierbar | Keine GUI, Lernkurve | Default für headless/CAN/IoT |

**Begründung für Wechsel**: Debian und Raspberry Pi wollen Konsistenz, Sicherheit und moderne Standards. *systemd-networkd* vermeidet Schwächen von *ifupdown* und passt zu IoT/Cloud-Trends Debian Release Notes systemd-networkd Docs.

# Was wird verwendet?

Um zu verstehen, was auf einem Raspberry Pi / SBC läuft, prüfe zuerst die installierten Pakete und Dienste – Bookworm mischt oft **NetworkManager** (GUI, WLAN/Ethernet) und **systemd-networkd** (headless, CAN). Schritte:

- **Pakete prüfen**: `dpkg -l | grep ifupdown` (zeigt, ob installiert; meist nicht in Bookworm).
- **Dienste prüfen**: `systemctl status systemd-networkd` (sollte “active” sein), `systemctl status NetworkManager` (aktiv für WLAN/Ethernet).
- **Interfaces analysieren**: `networkctl list` zeigt *systemd-networkd*-verwaltete Interfaces (z. B. *configured* für can0, *unmanaged* für andere). `nmcli device` zeigt *NetworkManager*-verwaltete (z. B. *connected* für wlan0). `ip link show` gibt Low-Level-Details (UP/DOWN, Queues). Konflikte (z. B. ein Interface in beiden) via Logs prüfen (siehe Kapitel 5) RPi Stack Exchange Debian Wiki.

```
networkctl list
IDX LINK TYPE OPERATIONAL SETUP
  1 lo   loopback carrier     unmanaged
  2 eth0 ether    no-carrier configuring
  3 can0 can     carrier     configured
  4 wlan0 wlan    routable    unmanaged
4 links listed.
```

```
nmcli device
DEVICE         TYPE      STATE            CONNECTION
wlan0          wifi      connected        FlyOS wireless
lo             loopback  connected (externally) lo
p2p-dev-wlan0  wifi-p2p  disconnected     --
eth0           ethernet  unavailable      --
can0           can       unmanaged        --
```

# CAN-Konfig

CAN-Interfaces (z. B. can0 für MCP2515-Chips, oder USB Koppler) sind von der Bookworm-Änderung betroffen, da sie früher über **ifupdown** liefen. Hier die Schritte für beide Methoden.

https://www.freedesktop.org/software/systemd/man/latest/networkd.conf.html
https://www.freedesktop.org/software/systemd/man/latest/systemd.network.html
https://www.freedesktop.org/software/systemd/man/latest/systemd.link.html

## Voraussetzungen

- Die CAN Hardware muss natürlich angeschlossen sein 😎
- CAB USB Adapter werden - wenn sich die richtige Firmware haben - in der Regel vom Kernel erkannt.
- SPI Devices brauchen oft eine extra Konfig. Beispiel MCP2515 am Raspberry Pi:
  - Editiere `/boot/config.txt`
  - `dtoverlay=mcp2515-can0,oscillator=8000000,interrupt=25`

## systemd-Methode (link)

- Konfigurationsdateien erstellen
  `sudo nano /etc/systemd/network/25-can.network`

```
[Match]
Name=can*
[CAN]
BitRate=1M
RestartSec=0.1s
[Link]
RequiredForOnline=no
```

- Konfigurationsdateien erstellen
  `sudo nano /etc/systemd/network/99-can-link.link`

```
[Match]
OriginalName=can*
[Link]
TransmitQueueLength=128
```

- Dienst aktivieren
  `sudo systemctl enable --now systemd-networkd`
- `sudo udevadm control --reload-rules && sudo udevadm trigger`
- Reboot erforderlich
  `sudo reboot`
- Prüfen
  - `ip link show can0` → Zeigt qlen 1024
  - `networkctl status can0` → Zeigt Bitrate 1M

### systemd-Methode (udev)

https://canbus.esoterical.online/Getting_Started.html

echo -e 'SUBSYSTEM==“net”, ACTION==“change|add”, KERNEL==“can*” ATTR{tx_queue_len}=“128”' | sudo tee /etc/udev/rules.d/10-can.rules > /dev/null

SUBSYSTEM==“net”, ACTION==“change|add”, KERNEL==“can*”, ATTR{tx_queue_len}=“128”

### ifupdown (legacy / alt)

Für Rückwärtskompatibilität. Schritte:

```
1. **Installieren**:
   ''code
   sudo apt install ifupdown
   ''
2. **Konfigurieren**: Erstelle ''/etc/network/interfaces.d/can0'':
   ''code
   auto can0
   iface can0 inet manual
   pre-up /sbin/ip link set can0 type can bitrate 1000000
   up /sbin/ifconfig can0 up
   post-up /sbin/ip link set can0 txqueuelen 1024
   down /sbin/ifconfig can0 down
   ''
3. **Aktivieren**:
   ''code
   sudo ifup can0
   ''
4. **Prüfen**:
   ''code
   ifconfig can0
   ip link show can0
   ''
```

**Nachteil**: Weniger integriert, potenziell langsamere Boots CANbus Guide.

### Migrationstipps

TBD Bei Upgrades von Bullseye: Entferne alte `/etc/network/interfaces`-Dateien, migriere zu `.network`-Format. Teste in einer VM, um Netzwerkbrüche zu vermeiden Medium Migration Story.

## nützliche Tools

Hier sind die wichtigsten Tools für Diagnose und Management, mit Beispielen und Anwendungsfällen:

- **Für systemd-networkd**:
  1. `networkctl list`: Übersicht aller Interfaces (z. B. can0 configured).
  2. `networkctl status <iface>`: Details zu Bitrate/Status (z. B. can0 Bitrate).
  3. `systemd-analyze`: Boot-Zeiten-Analyse für Netzwerk-Verzögerungen.
- **Für NetworkManager**:
  1. `nmcli device`: Geräte-Status (z. B. wlan0 connected).
  2. `nmcli connection show`: Verbindungs-Details (IPs, DNS).
  3. `nmtui`: Text-UI für headless Setups.
  4. `nm-connection-editor`: GUI für Desktop.
- **Allgemein**:
  1. `ip link show <iface>`: Low-Level-Status, Queues.
  2. `ip addr show <iface>`: IPs und Adressen.
  3. `ls /etc/systemd/network/`: Configs anzeigen networkctl Docs NM Docs Arch Wiki.

**Demo**: Führe `nmcli connection add` und `networkctl status can0` live aus.

## journalctl / dmesg

Debugging ist essenziell für Netzwerkprobleme. Hier die wichtigsten Log-Tools:

- **journalctl**: Systemd-Logs. Nutze:
  1. `journalctl -u systemd-networkd`: Logs für *systemd-networkd* (suche “Link UP”, Errors wie “Failed to bring up”).
  2. `journalctl -u NetworkManager`: Logs für NM.
  3. **Achten auf**: Timestamps (`-S “5 minutes ago”`), Prioritäten (`-p err` für Errors), Units (`-u`). Live-Monitoring mit `–follow`.
- **dmesg**: Kernel-Logs. Nutze:
  1. `dmesg | grep can`: CAN-Treiber-Init (z. B. “MCP2515 initialized” oder “No carrier”).
  2. **Achten auf**: Zeitstempel (seit Boot), Level (`-l err` für Errors).
- **Andere Stellen**:
  1. `/var/log/syslog`: Allgemeine Logs.
  2. `/var/log/kern.log`: Kernel-Logs.
  3. `systemctl status <dienst>`: Kurze Übersicht.
  4. `journalctl -b`: Logs seit letztem Boot journalctl Docs dmesg Docs Debian Journal.

**Demo**: Simuliere einen Error (z. B. falsche Bitrate), zeige Logs: `code journalctl -u systemd-networkd -p err dmesg | grep can`

**Tipp**: `journalctl –vacuum-time=2weeks` zum Log-Aufräumen.

## Parallelbetrieb NM & systemd-networkd

In Bookworm laufen **NetworkManager** (WLAN/Ethernet) und **systemd-networkd** (headless/CAN) oft parallel, was Konflikte verursachen kann. Häufige Probleme:

- **Konflikte bei Interfaces**: Z. B. eth0 als *configuring* in `networkctl`, aber *unavailable* in `nmcli` – beide Manager greifen zu.
- **Symptome**: Doppelte IPs, Boot-Verzögerungen (z. B. `systemd-networkd-wait-online` fail), Interfaces nicht erreichbar.
- **Ursachen**: Überlappende Konfigurationen (z. B. `.network`-Datei für eth0 in `/etc/systemd/network/`). Community-Berichte über Netzwerkbrüche bei Upgrades Medium Story RPi Forum Konflikte.

**Lösungen**:

- **Trenne Interfaces**: In `/etc/NetworkManager/NetworkManager.conf`:

`code [keyfile] unmanaged-devices=interface-name:can*`

```
  Danach: ''sudo systemctl restart NetworkManager''.
* **Deaktiviere einen Manager**: Z. B. ''sudo systemctl disable
NetworkManager''.
* **Prüfe Konfig-Dateien**: Entferne eth0 aus ''/etc/systemd/network/'':
  ''code
  sudo rm /etc/systemd/network/<eth0-datei>.network
  sudo systemctl restart systemd-networkd
  ''
```

**Demo**: Zeige Konflikt aus Rechner-Daten, dann Fix: `code nmcli connection add type ethernet ifname eth0 con-name “Wired connection 1”`

## CAN mit NM?

Kurz: Möglich, aber **nicht empfohlen**. **NetworkManager** ist für IP-basierte Netzwerke (WiFi, Ethernet, VPN) optimiert und unterstützt CAN nicht nativ (keine Bitrate/Queue-Optionen). Workaround: Custom-Skripte in `/etc/NetworkManager/dispatcher.d/` (z. B. für `ip link set can0 type can bitrate 1M`), aber das ist kompliziert und unstabil. Community empfiehlt **systemd-networkd** für CAN, da es low-level-Protokolle besser handhabt. NM sollte CAN als *unmanaged* markieren Stack Exchange NM Config Docs.

**Demo**: Zeige Fehlversuch mit `nmcli` (kein CAN-Support), dann Workaround: `code sudo ip link set can0 up type can bitrate 1000000`

**Tipp**: Bleib bei *systemd-networkd* für CAN-Stabilität.

# Befehlsreferenz

Hier eine Zusammenfassung aller nützlichen Befehle aus dem Video als Quick-Reference:

| Kategorie | Befehl | Beschreibung & Beispiel |
|---|---|---|
| **Installation/Status** | `dpkg -l \| grep ifupdown` | Prüft, ob ifupdown installiert ist. |
| | `systemctl status systemd-networkd` | Zeigt Status von systemd-networkd (active/inactive). |
| | `systemctl status NetworkManager` | Zeigt Status von NM. |
| **Interface-Übersicht** | `networkctl list` | Listet systemd-verwaltete Interfaces (z. B. can0 configured). |
| | `nmcli device` | Listet NM-verwaltete Interfaces (z. B. wlan0 connected). |
| | `ip link show <iface>` | Low-Level-Details (z. B. `ip link show can0` für Queue/Status). |
| | `ip addr show <iface>` | IPs und Adressen (z. B. für eth0). |
| **Konfiguration** | `sudo systemctl enable --now systemd-networkd` | Aktiviert systemd-networkd. |
| | `sudo ifup <iface>` | Aktiviert Interface mit ifupdown (z. B. can0). |
| | `nmcli connection add type ethernet ifname eth0 con-name “Wired”` | Fügt NM-Verbindung hinzu. |
| | `sudo udevadm control --reload-rules && sudo udevadm trigger` | Lädt Udev-Regeln neu (für .link-Dateien). |
| **Debugging** | `journalctl -u systemd-networkd` | Logs für systemd-networkd (filtere mit -p err). |
| | `journalctl -u NetworkManager` | Logs für NM. |
| | `dmesg \| grep can` | Kernel-Logs für CAN (z. B. Treiber-Init). |
| | `systemd-analyze blame` | Boot-Zeiten-Analyse (zeigt Netzwerk-Verzögerungen). |
| **Manuell Aktivieren** | `sudo ip link set can0 up type can bitrate 1M` | Manuelles CAN-Setup (Fallback). |

# Python Testscript

DiesesPython Script zeigt, welches Interface von welchem Manager gemanaged wird.

Beispielausgabe

```
fly@fly-minipad:~$ python3 can.py
System Information:
OS                  : Armbian 25.8.1 bookworm
Kernel              : 5.10.85-v3.0-fly-sunxi

Interfaces and Managers:
```

```
+---------------+----------------------+---------------------------------+
| Interface     | Manager              | Configuration Source            |
+---------------+----------------------+---------------------------------+
| can           | ifupdown             | /etc/network/interfaces.d/can1  |
| can0          | ifupdown             | /etc/network/interfaces.d/can0  |
| can1          | ifupdown             | /etc/network/interfaces.d/can1  |
| eth0          | Unmanaged or Unknown | None                            |
| eth1          | NetworkManager       | Connection: Wired               |
| inet          | ifupdown             | /etc/network/interfaces         |
| lo            | NetworkManager       | Connection: (externally)        |
| p2p-dev-wlan0 | Unmanaged or Unknown | None                            |
| wlan0         | NetworkManager       | Connection: NoMamsLand          |
+---------------+----------------------+---------------------------------+
```

can.py

```
#!/usr/bin/env python3
import subprocess
import re
import os
import glob
from platform import release, system, uname

def run_command(cmd):
    result = subprocess.run(cmd, shell=True, capture_output=True,
text=True)
    return result.stdout.splitlines()

def parse_networkctl():
    networkctl = run_command("networkctl list")
    interfaces = {}
    for line in networkctl[1:]:  # Skip header
        parts = re.split(r'\s+', line.strip())
        if len(parts) >= 5:
            iface, setup = parts[1], parts[-1]
            interfaces[iface] = {"setup": setup, "config": None}
            for f in glob.glob("/etc/systemd/network/*.network"):
                with open(f, "r") as file:
                    if f"Name={iface}" in file.read() or "Name=*" in
file.read():
                        interfaces[iface]["config"] = f
    return interfaces

def parse_nmcli():
    nmcli = run_command("nmcli device")
    interfaces = {}
    for line in nmcli[1:]:  # Skip header
        parts = re.split(r'\s+', line.strip())
        if len(parts) >= 4:
```

```
            iface, state, conn = parts[0], parts[2], parts[3]
            interfaces[iface] = {"state": state, "config":
f"Connection: {conn}" if conn != "--" else None}
    return interfaces

def check_ifupdown():
    interfaces = {}
    if os.path.exists("/etc/network/interfaces"):
        with open("/etc/network/interfaces", "r") as f:
            for line in f:
                if line.strip().startswith(("iface ", "allow-hotplug
")):
                    parts = line.split()
                    if len(parts) > 1:
                        iface = parts[1] if parts[0] == "allow-hotplug"
else parts[2]
                        interfaces[iface] = {"manager": "ifupdown",
"config": "/etc/network/interfaces"}
    for file in glob.glob("/etc/network/interfaces.d/*"):
        with open(file, "r") as f:
            for line in f:
                if line.strip().startswith(("iface ", "allow-hotplug
")):
                    parts = line.split()
                    if len(parts) > 1:
                        iface = parts[1] if parts[0] == "allow-hotplug"
else parts[2]
                        interfaces[iface] = {"manager": "ifupdown",
"config": file}
    return interfaces

def print_system_info():
    os_info = "Raspberry Pi OS" if os.path.exists("/etc/rpi-issue")
else "Unknown OS"
    with open("/etc/os-release", "r") as f:
        for line in f:
            if line.startswith("PRETTY_NAME="):
                os_info = line.split("=")[1].strip().strip('"')
                break
    kernel = release()
    print(f"System Information:")
    print(f"{'OS':<20}: {os_info}")
    print(f"{'Kernel':<20}: {kernel}")
    print()

def print_table(data):
    headers = ["Interface", "Manager", "Configuration Source"]
    max_len = [len(h) for h in headers]
    rows = []
    for iface, info in data.items():
        manager = info.get("manager", "Unmanaged or Unknown")
```

```
        config = info.get("config", "None")
        rows.append([iface, manager, config])
        max_len = [max(max_len[i], len(str(col))) for i, col in
enumerate([iface, manager, config])]

    print("+" + "+".join("-" * (l + 2) for l in max_len) + "+")
    print("|" + "|".join(f" {h:<{max_len[i]}} " for i, h in
enumerate(headers)) + "|")
    print("+" + "+".join("-" * (l + 2) for l in max_len) + "+")

    for row in sorted(rows, key=lambda x: x[0]):
        print("|" + "|".join(f" {col:<{max_len[i]}} " for i, col in
enumerate(row)) + "|")
    print("+" + "+".join("-" * (l + 2) for l in max_len) + "+")

def main():
    print_system_info()
    print("Interfaces and Managers:")

    networkctl_ifaces = parse_networkctl()
    nmcli_ifaces = parse_nmcli()
    ifupdown_ifaces = check_ifupdown()

    all_ifaces = set(networkctl_ifaces.keys()) |
set(nmcli_ifaces.keys()) | set(ifupdown_ifaces.keys())
    result = {}

    for iface in all_ifaces:
        if iface in ifupdown_ifaces:
            result[iface] = {"manager": "ifupdown", "config":
ifupdown_ifaces[iface]["config"]}
        elif networkctl_ifaces.get(iface, {}).get("setup", "") in
["configured", "configuring"]:
            result[iface] = {"manager": "systemd-networkd", "config":
networkctl_ifaces[iface]["config"]}
        elif nmcli_ifaces.get(iface, {}).get("state", "") in
["connected", "connecting"]:
            result[iface] = {"manager": "NetworkManager", "config":
nmcli_ifaces[iface]["config"]}
        else:
            result[iface] = {"manager": "Unmanaged or Unknown",
"config": "None"}

    print_table(result)

if __name__ == "__main__":
    main()
```

# Links

Weitere Themen: Vorteile von **systemd-networkd** (automatische Restarts), Nachteile (keine GUI), Alternativen wie Netplan (Ubuntu-fokussiert). Integration mit raspi-config für WiFi, CAN in Docker.

- Raspberry Pi Foren: Bookworm Änderungen, NM vs. systemd, CAN Setup
- Debian Wiki: systemd-networkd Guide
- Stack Exchange: systemd Nutzung
- Medium: Migration Story
- Arch Wiki: Erweiterte Docs
- Hackaday: RPi Netzwerk
- Offizielle RPi-Docs: Netzwerk-Konfig