TECHNISCHE UNIVERSITÄT DRESDEN

DRESDEN concept

Lucas Waclawczyk

TUN/TAP-Geräte
Übersicht, Funktionsweise und Implementierung im Linux-Kernel

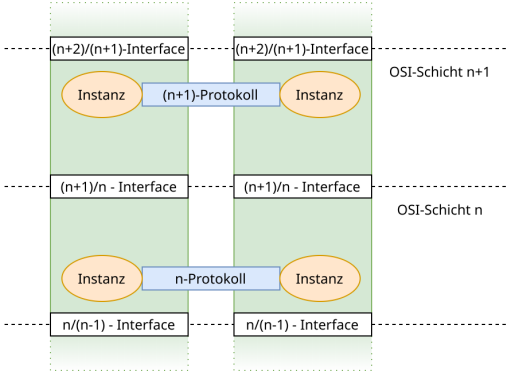
Proseminar Rechnernetze // Dresden, 18. Juni 2020

Inhalt

1. Übersicht

Network Interfaces (sh. [1])

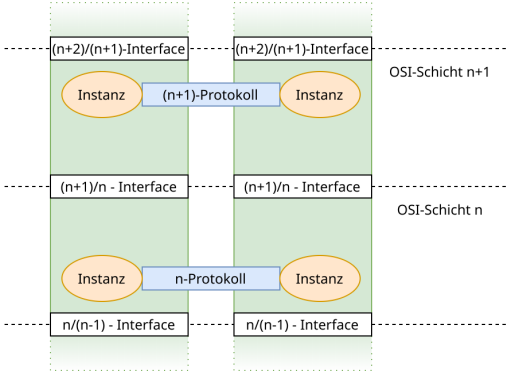
- Interface = Schnittstelle
- zwischen zwei Schichten im OSI-Modell



Ausschnitt des OSI-Schichtenmodells, entlehnt [1]

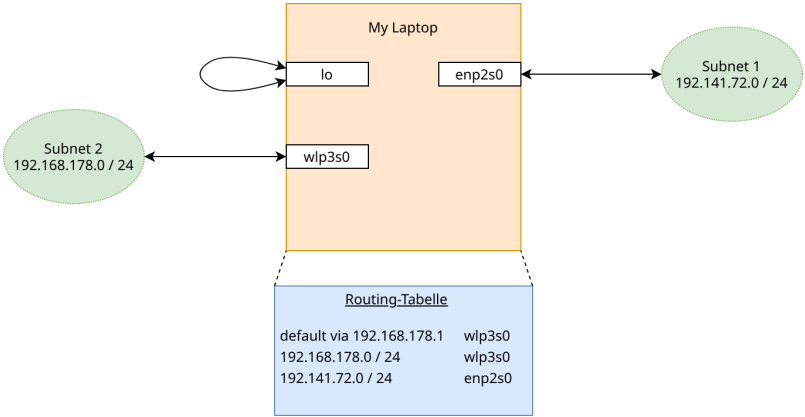
Network Interfaces (sh. [1])

- Interface = Schnittstelle
- zwischen zwei Schichten im OSI-Modell
- übernimmt Daten von Instanz eines $(n+1)$-Protokolls
- stellt Daten für Instanz eines (n)-Protokolls bereit



Ausschnitt des OSI-Schichtenmodells, entlehnt [1]

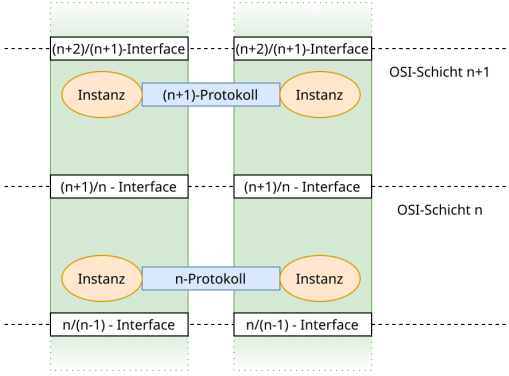
Network Interfaces (sh. [1])



Network Interfaces relativ zu Gerät und Netz am Beispiel

Network Interfaces (sh. [1])

- Interface = Schnittstelle
- zwischen zwei Schichten im OSI-Modell
- übernimmt Daten von Instanz eines $(n+1)$-Protokolls
- stellt Daten für Instanz eines (n)-Protokolls bereit
- muss nicht physisch sein
  ➔ Virtual Network Interface



Ausschnitt des OSI-Schichtenmodells, entlehnt [1]

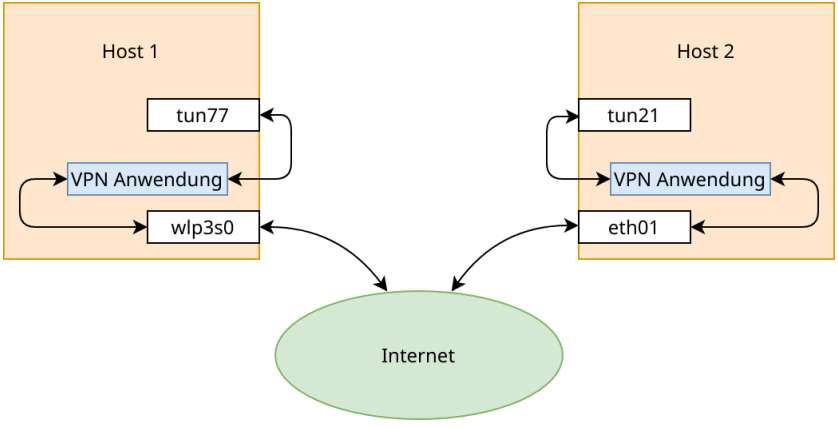
Generelles zu TUN und TAP

- Virtual Network Interfaces, d.h. man kann
  - … ihnen IP-Adressen zuweisen
  - … ihren Traffic analysieren
  - … Firewall-Regeln für sie konfigurieren
  - … uvm.

Generelles zu TUN und TAP

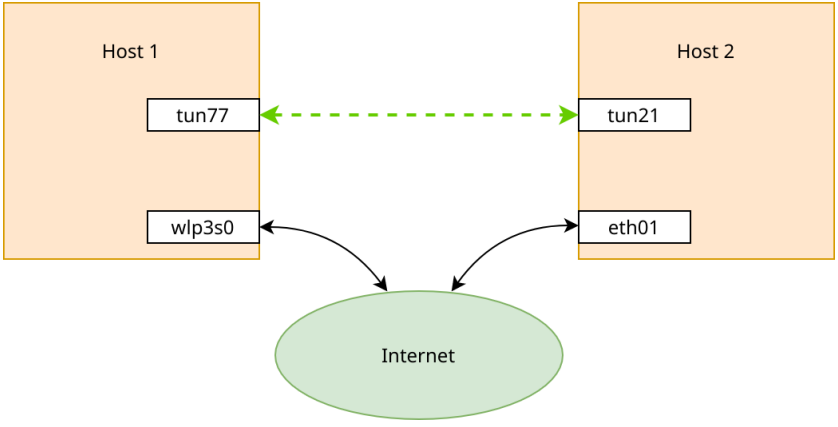
- Virtual Network Interfaces, d.h. man kann
  - … ihnen IP-Adressen zuweisen
  - … ihren Traffic analysieren
  - … Firewall-Regeln für sie konfigurieren
  - … uvm.
- *Interface* $\leftrightarrow$ *Anwendung* statt *Interface* $\leftrightarrow$ *physische Verbindung*
- nützlich für virtuelle Verbindungen

Generelles zu TUN und TAP



Schema einer virtuellen Verbindung (physische Ansicht)

Generelles zu TUN und TAP



Schema einer virtuellen Verbindung (virtuelle Ansicht)

Generelles zu TUN und TAP

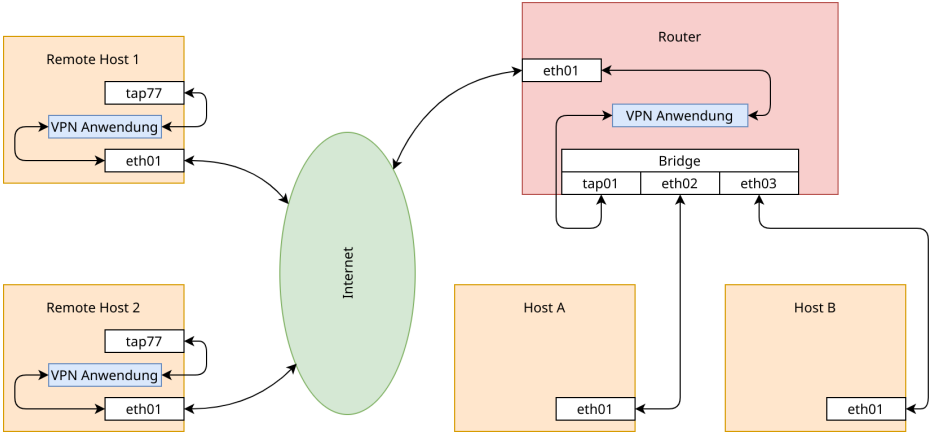
- Virtual Network Interfaces, d.h. man kann
  - … ihnen IP-Adressen zuweisen
  - … ihren Traffic analysieren
  - … Firewall-Regeln für sie konfigurieren
  - … uvm.
- *Interface* $\leftrightarrow$ *Anwendung* statt *Interface* $\leftrightarrow$ *physische Verbindung*
- nützlich für virtuelle Verbindungen
- u. A. Linux, Windows 2000 - 10, Mac OS X (nur TUN eingebaut)
- hier für Linux

TUN vs. TAP (sh. [7])

TAP – Terminal Access Point
$\sim$ Ethernet $\sim$ Layer 2

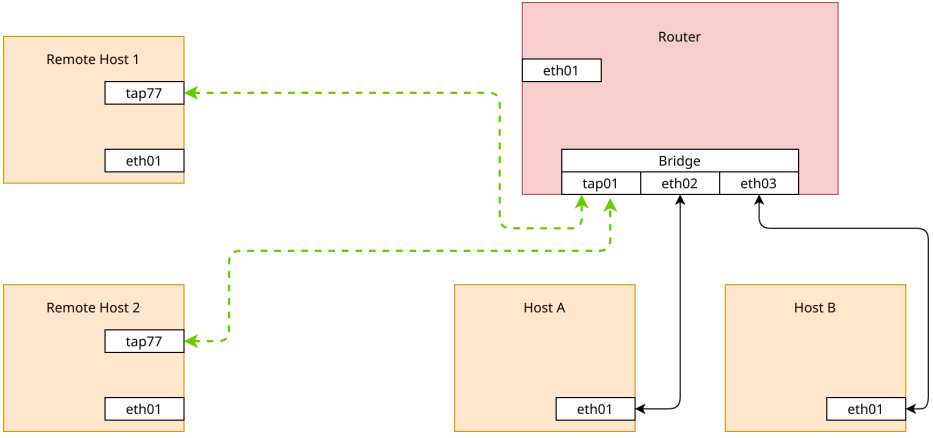
TUN – Netzwerk-Tunnel
$\sim$ IP $\sim$ Layer 3

TUN vs. TAP (sh. [7])



Anwendungsbeispiel TAP (physische Ansicht)

TUN vs. TAP (sh. [7])



Anwendungsbeispiel TAP (virtuelle Ansicht)

TUN vs. TAP (sh. [7])

TAP – Terminal Access Point

$\sim$ Ethernet $\sim$ Layer 2

- $\oplus$ verhält sich wie echter Netzwerkadapter
- $\oplus$ flexible Protokollwahl
- $\oplus$ Bridging möglich

TUN – Netzwerk-Tunnel

$\sim$ IP $\sim$ Layer 3

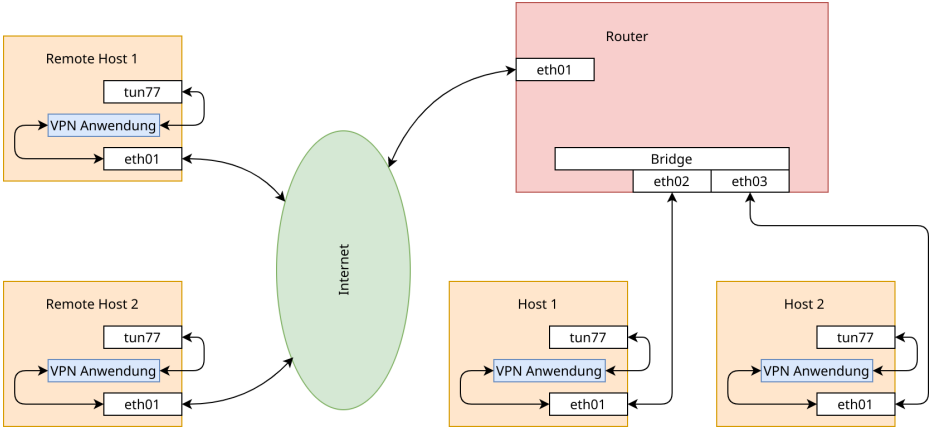
TUN vs. TAP (sh. [7])

TAP – Terminal Access Point
$\sim$ Ethernet $\sim$ Layer 2

- $\oplus$ verhält sich wie echter Netzwerkadapter
- $\oplus$ flexible Protokollwahl
- $\oplus$ Bridging möglich
- $\ominus$ Overhead (Ethernet-Header, Broadcast)
- $\ominus$ skaliert schlechter als TUN
- $\ominus$ kein Support bei Android, iOS

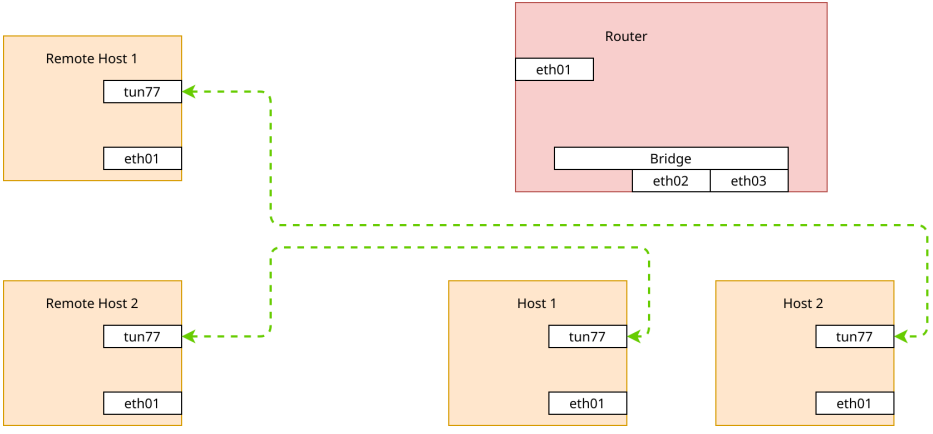
TUN – Netzwerk-Tunnel
$\sim$ IP $\sim$ Layer 3

TUN vs. TAP (sh. [7])



Anwendungsbeispiel TUN (physische Ansicht)

TUN vs. TAP (sh. [7])



Anwendungsbeispiel TUN (virtuelle Ansicht)

TUN vs. TAP (sh. [7])

TAP – Terminal Access Point
$\sim$ Ethernet $\sim$ Layer 2

- $\oplus$ verhält sich wie echter Netzwerkadapter
- $\oplus$ flexible Protokollwahl
- $\oplus$ Bridging möglich
- $\ominus$ Overhead (Ethernet-Header, Broadcast)
- $\ominus$ skaliert schlechter als TUN
- $\ominus$ kein Support bei Android, iOS

TUN – Netzwerk-Tunnel
$\sim$ IP $\sim$ Layer 3

- $\oplus$ weniger Overhead (kein Ethernet-Header, kein Broadcast)
- $\oplus$ nur Layer-3-Pakete

TUN vs. TAP (sh. [7])

TAP – Terminal Access Point
∼ Ethernet ∼ Layer 2

- ⊕ verhält sich wie echter Netzwerkadapter
- ⊕ flexible Protokollwahl
- ⊕ Bridging möglich
- ⊖ Overhead (Ethernet-Header, Broadcast)
- ⊖ skaliert schlechter als TUN
- ⊖ kein Support bei Android, iOS

TUN – Netzwerk-Tunnel
∼ IP ∼ Layer 3

- ⊕ weniger Overhead (kein Ethernet-Header, kein Broadcast)
- ⊕ nur Layer-3-Pakete
- ⊖ nur Layer-3-Pakete
- ⊖ keine Broadcasts
- ⊖ kein Bridging möglich

2. Implementierung in Linux

Wichtige Adressen

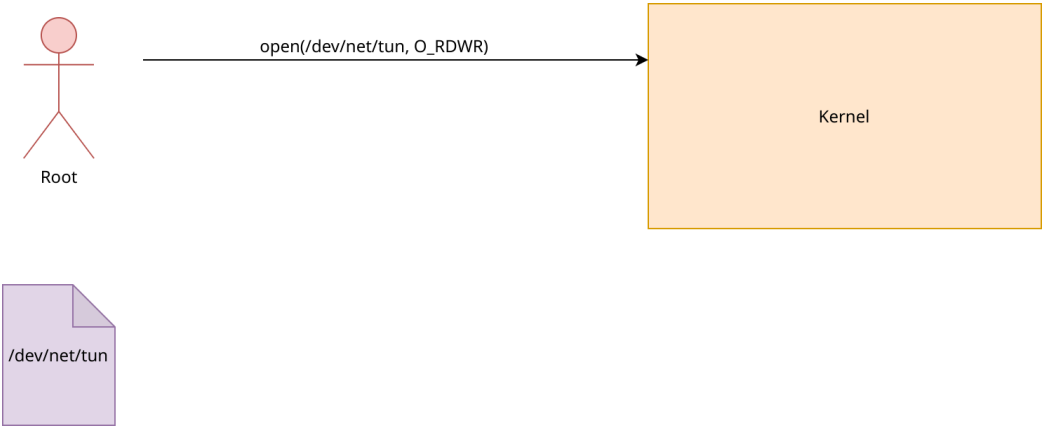
- Kernel-Code:
  `https://github.com/torvalds/linux.git`
- TUN / TAP (Driver):
  `/linux/drivers/net/tun.c`
- Clone Device:
  `/dev/net/tun`
- diese Präsentation und Code:
  `https://github.com/lucaswzyk/pres_tun_tap.git`

Aufbau (sh. [2])

Aufbau eines Interfaces (erfordert CAP_NET_ADMIN capability):

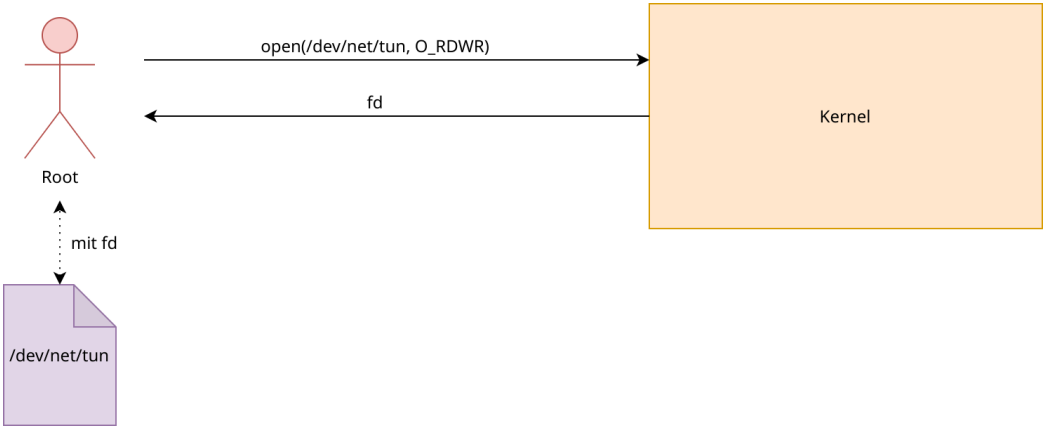
1. öffne `/dev/net/tun` (Clone Device) mit Lese-Schreibberechtigung
2. rufe syscall `ioctl(fd, TUNSETIFF, if_req_struct)` auf

Aufbau (sh. [2])



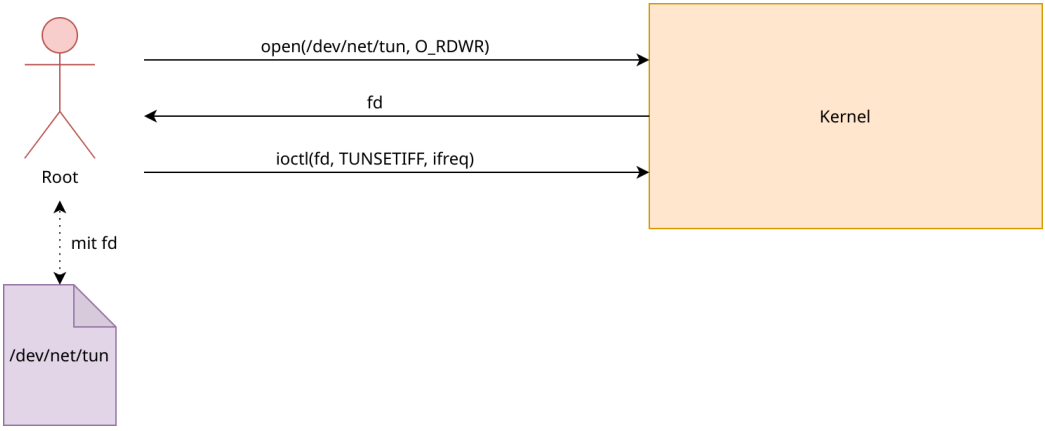
Aufbau eines Virtual Network Interfaces (1 / 4)

Aufbau (sh. [2])



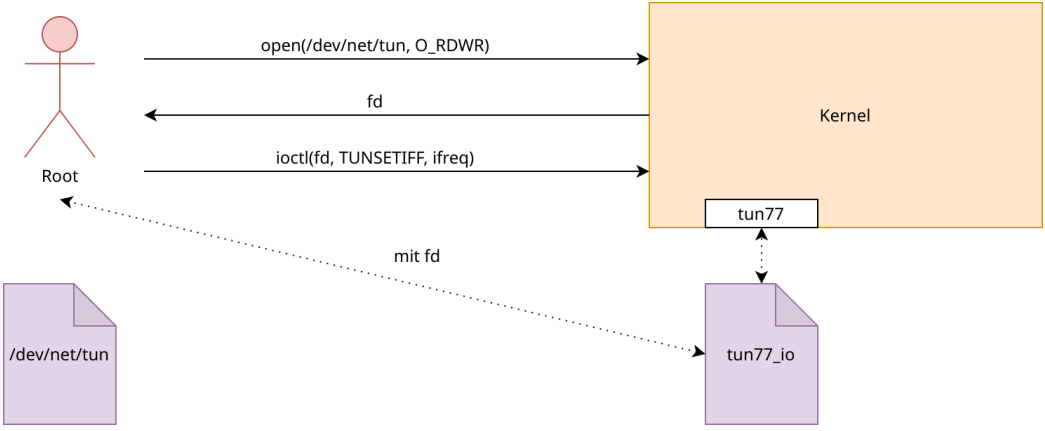
Aufbau eines Virtual Network Interfaces (2 / 4)

Aufbau (sh. [2])



Aufbau eines Virtual Network Interfaces (3 / 4)

Aufbau (sh. [2])



Aufbau eines Virtual Network Interfaces (4 / 4)

Aufbau (sh. [2])

Aufbau eines Interfaces (erfordert CAP_NET_ADMIN capability):

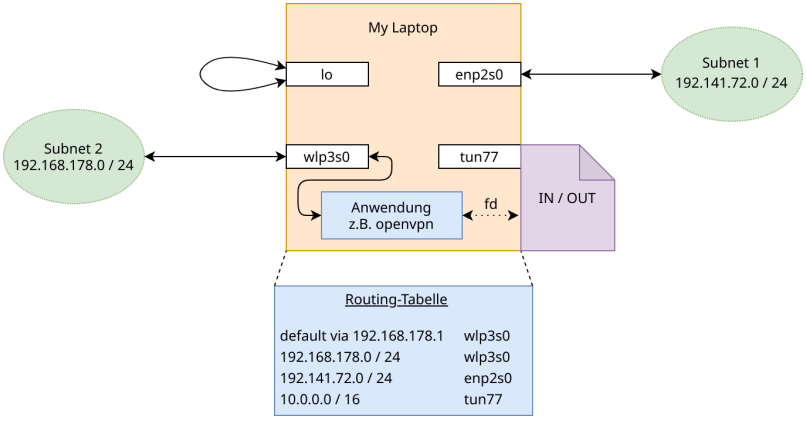
1. öffne `/dev/net/tun` (Clone Device) mit Lese-Schreibberechtigung
2. rufe syscall `ioctl(fd, TUNSETIFF, if_req_struct)` auf
3. persistiere Gerät (ggf.)
4. weise IP-Adresse(n) zu

Nutzung und Abbau (sh. [2])

Nutzung:

- binde Anwendung an Interface
  - wiederhole obiges als beliebiger Nutzer
  - nutze bestehenden Interface-Namen
- Lesen und Schreiben mittels `fd`

Nutzung und Abbau (sh. [2])



Network Interfaces (physisch und virtuell) relativ zu Gerät und Netz am Beispiel

Nutzung und Abbau (sh. [2])

Nutzung:

- binde Anwendung an Interface
  - wiederhole obiges als beliebiger Nutzer
  - nutze bestehenden Interface-Namen
- Lesen und Schreiben mittels `fd`

Abbau:

- transientes Gerät verschwindet mit Beenden des Erzeuger-Prozesses
- persistentes Gerät muss aktiv abgebaut werden (syscall)

Umsetzung als Structs

Auszug aus `/linux/drivers/net/tun.c`:

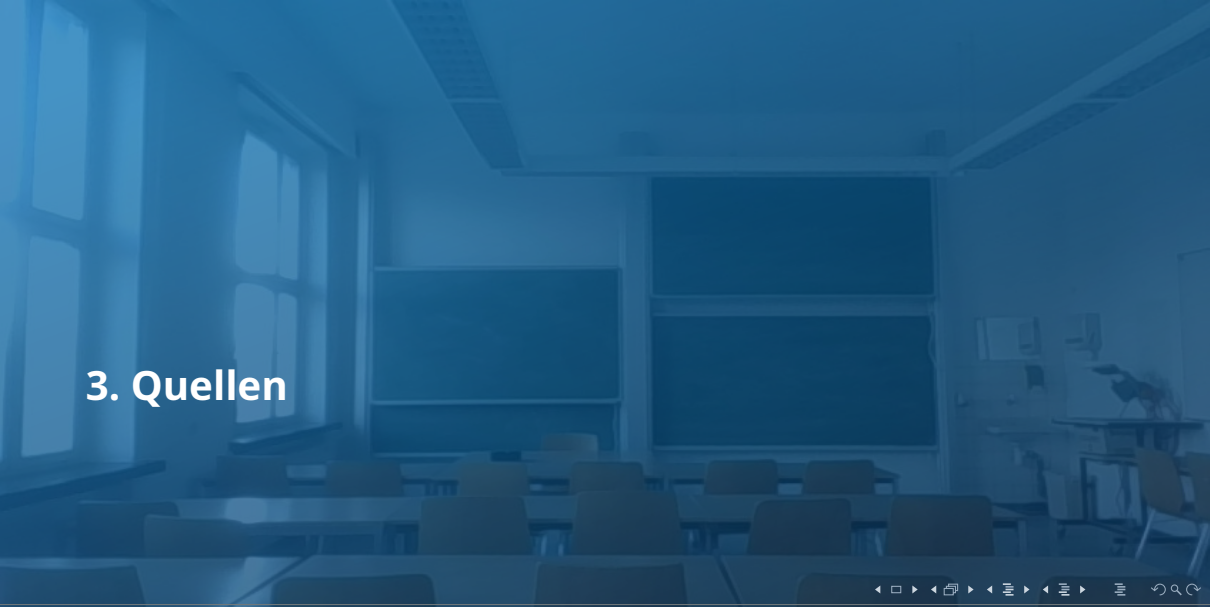
```
struct tun_struct {
        struct tun_file __rcu *tfiles[MAX_TAP_QUEUES];
        unsigned int flags;
        kuid_t owner;
        kuid_t group;
        struct net_device *dev;
}

struct tun_file {
        struct tun_struct __rcu *tun;
}
```

Simples Code-Beispiel

- Erstellen eines TUN-Devices mit openvpn und Anbinden einer simplen Anwendung
- Erläuterung und Bilder im Repo

Simples Code-Beispiel

- Erstellen eines TUN-Devices mit openvpn und Anbinden einer simplen Anwendung
- Erläuterung und Bilder im Repo
- Viel Spaß beim Probieren und vielen Dank!



3. Quellen

Quellen

[1] Skript und Übungsaufgaben der Vorlesung Rechnernetze, TU Dresden 2019

[2] https://backreference.org/2010/03/26/tuntap-interface-tutorial/

[3] https://www.elektronik-kompendium.de/sites/net/0811011.htm

[4] https://en.wikipedia.org/wiki/TUN/TAP

[5] https://www.thomas-krenn.com/de/wiki/OpenVPN_Grundlagen

[6] https://floating.io/2016/05/tuntap-demystified/3/

[7] https://community.openvpn.net/openvpn/wiki/BridgingAndRouting

[8] Grafiken wurden erstellt auf draw.io