Hochschule für Technik
und Wirtschaft Berlin

University of Applied Sciences

Fachbereich 4 Wirtschaftswissenschaften II

Laborprotokoll

Netzwerke Übung
Sommersemester 2019

im Studiengang Angewandte Informatik (B.Sc.)

Thema: Übungsblatt 02 – Netzwerkgrundlagen
`Beispiellösung`

eingereicht von:

Name1	name1@student.htw-berlin.de	matrikel1
Name2	name2@student.htw-berlin.de	matrikel2
Name3	name3@student.htw-berlin.de	matrikel3
Name4	name4@student.htw-berlin.de	matrikel4

eingereicht am: 7. Mai 2019

Inhaltsverzeichnis

1. Einleitung

Ziel des Laborprotokolls ist es Lösungswege aufzuzeigen, die es erlauben bestehende Netzwerkkonfigurationen zu analysieren und ein einfaches geswitchtes Local Area Network (LAN) zu realisieren.

1.1. Aufgaben- & Problemstellung

In der zweiten Laborübung haben wir uns erstmals mit der Konfiguration von Netzwerkadaptern auseinandergesetzt. Die dafür notwendigen Komponenten und das genutzte Betriebssystem werden im Abschnitt 2.1 in Kürze beschrieben.

Zunächst untersuchen wir die Infrastruktur mithilfe des durch das *Dynamic Host Configuration Protocol (DHCP)* vorkonfigurierten Netzwerkes. Dies repräsentiert den ersten Aufgabenteil und wird im Abschnitt 2.2 beschrieben. Anschließend setzen wir ein eigenes statisches *LAN* um, das aus der Planung der Hausaufgaben hervorgeht. Die Protokollierung des geswitchten *LAN*s ist im Abschnitt 2.3 festgehalten.

2. Hauptteil

2.1. Raspberry Pi

Die in der Laborübung genutzen Raspberry Pi 3 b+ sind sogenannte Systems-on-a-Chip (SoCs), die als vollständige Rechner gelten (*Turing-vollständig*). Näheres zur Hardware kann unter [Fou19] gefunden werden.
Wir arbeiten auf einem GNU/Linux System mit einer Raspbian Distribution (Debian-Fork) wie im Listing 2.1 zu sehen ist.

```
cat /etc/os-release
NAME="Arch Linux"
PRETTY_NAME="Arch Linux"
ID=arch
ID_LIKE=archlinux
ANSI_COLOR="0;36"
HOME_URL="https://www.archlinux.org/"
SUPPORT_URL="https://bbs.archlinux.org/"
BUG_REPORT_URL="https://bugs.archlinux.org/"
```

`uname` gibt folgende Kernelspezifika aus:

```
uname -or
5.0.10-hardend-arch1-1-ARCH GNU/Linux # Linux Kernel 5.0.101 modifizierter, gehaerteten Kern
# RPI
uname -or
4.14.98-v7+ GNU/Linux
```

Der Hostname alles Raspberry Pis lautet *networkingLab*, auf diesen Rechnern arbeiten wir als Nutzer *student* und befinden uns standardmäßig zunächst im Heimatverzeichnis – s. Listing 2.1.

```
1 whoami
2 student
3 pwd
4 /home/student
5 hostname
6 networkingLab
```

Um zu überprüfen, ob das *DHCP* eingeschaltet ist, verwenden wir das Systemd-Werkzeug *systemctl*. Mithilfe von *systemctl* können die Stati der Dienste eines Betriebssystems abgefragt und neu gesetzt werden. Es ist somit Administrationswerkzeug für *Daemons*. Eine ausführliche Beschreibung von *systemd* und *systemctl* kann unter [Wik19a] oder [Wik19b] gefunden werden.
Das im Listing 2.1 ausgeführte Kommando zeigt, dass das *DHCP* eingeschaltet ist.

```
1  sudo systemctl status dhcpcd
2  dhcpcd.service -dhcpcd on all interfaces
3     Loaded: loaded (/lib/systemd/system/dhcpcd.service; enabled; vendor preset: enabled)
4    Drop-In: /etc/systemd/system/dhcpcd.service.d
5             wait.conf
6     Active: active (running) since Sun 2019-05-05 10:41:46 CEST; 3min 27s ago
7    Process: 355 ExecStart=/usr/lib/dhcpcd5/dhcpcd -q -w (code=exited, status=0/SUCCESS)
8   Main PID: 590 (dhcpcd)
9     CGroup: /system.slice/dhcpcd.service
10            422 wpa_supplicant -B -c/etc/wpa_supplicant/wpa_supplicant.conf -iwlan0 -Dnl80211,wext
11            590 /sbin/dhcpcd -q -w
12
13 May 05 10:41:44 networkingLab dhcpcd[355]: eth0: adding route to 2a02:8109:83c0:371e::/64
14 May 05 10:41:44 networkingLab dhcpcd[355]: eth0: adding default route via fe80::e228:6dff:fe54:68bf
15 May 05 10:41:44 networkingLab dhcpcd[355]: eth0: requesting DHCPv6 information
16 May 05 10:41:45 networkingLab dhcpcd[355]: Too few arguments.
17 May 05 10:41:45 networkingLab dhcpcd[355]: Too few arguments.
18 May 05 10:41:46 networkingLab dhcpcd[355]: forked to background, child pid 590
19 May 05 10:41:46 networkingLab systemd[1]: Started dhcpcd on all interfaces.
20 May 05 10:41:48 networkingLab dhcpcd[590]: eth0: leased 10.10.10.228 for 864000 seconds
21 May 05 10:41:48 networkingLab dhcpcd[590]: eth0: adding route to 10.10.10.254/24
22 May 05 10:41:48 networkingLab dhcpcd[590]: eth0: adding default route via 10.10.10.254
```

Als Vorgriff auf den nächsten Abschnitt sind alle wichtigen Informationen bereits in dieser Statusabfrage gegeben.
Die Einträge des Listings 2.1 sind dergestalt, das zuerst beschrieben wird, wann welche Konfiguration vorgenommen wurde. Anschließend erfolgt der Eintrag auf welchem System (*Hostname*) und welcher Dienst (*Daemon*). Daran anschließend sind in diesem Fall die Netzwerkdetails zu sehen.
In Zeile 6 ist zu lesen, dass der *dhcpcd*-Dienst aktiviert ist und welchen Process Identifier (PID) dieser Prozess im Betriebssystem zugeordnet ist. In den Zeilen 13-15 werden

die Routing-Informationen für *IPv6* angezeigt. Die Zeilen 20-22 zeigen die äquivalente Konfiguration für das *IPv4*-Protokoll. Ebenfalls abzulesen ist, dass ausschließlich das Interface `eth0` via *dhcpcd* konfiguriert wurde. Auf dem Interface `eth0` wurde eine *IPv4*-Adresse durch den *DHCP* konfiguriert.[1] Weiterhin zu sehen sind, dass die Routen für *IPv6* und *IPv4* gesetzt wurden, sodass eine Kommunikation auch außerhalb des *LAN*s möglich ist.

2.2. Analyse bestehender Netzwerkkonfiguration

Wie bereits im einleitenden Abschnitt beschrieben, soll hier die Analyse eines bereits vorkonfigurierten Netzwerkadapters erfolgen.
Zunächst sollte die aktuelle *IP*-Adresskonfiguration in Erfahrung gebracht werden. Dies kann mithilfe der Tools *ip* oder *ifconfig* geschehen.

Im Allgemeinen kann auf der Kommandozeile wie folgt vorgegangen werden:

```
ip addr  #zeigt alle geraete
ip addr show eth0 # zeigt nur konfig. von eth0
ifconfig # zeigt alle geraete
ifconfig eth0 # zeigt konfig. von eth0
```

Die beiden nachfolgenden Listings (s. 2.2, 2.2) sind zwei Ausgaben exemplarisch für das Interface `eth0` gegeben.

```
ip addr show eth0
2: eth0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc pfifo_fast state UP group default qlen 1000
    link/ether b8:27:eb:21:a0:54 brd ff:ff:ff:ff:ff:ff
    inet 10.10.10.228/24 brd 192.168.178.255 scope global eth0
       valid_lft forever preferred_lft forever
    inet6 2a02:8109:83c0:371e:54d7:4903:af2e:386f/64 scope global mngtmpaddr noprefixroute dynamic
       valid_lft 4559sec preferred_lft 1859sec
    inet6 fe80::67a1:ecfc:4dd8:bc0d/64 scope link
       valid_lft forever preferred_lft forever
```

```
ifconfig eth0
eth0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST> mtu 1500
        inet 10.10.10.228 netmask 255.255.255.0 broadcast 192.168.178.255
        inet6 2a02:8109:83c0:371e:54d7:4903:af2e:386f prefixlen 64 scopeid 0x0<global>
        inet6 fe80::67a1:ecfc:4dd8:bc0d prefixlen 64 scopeid 0x20<link>
        ether b8:27:eb:21:a0:54 txqueuelen 1000 (Ethernet)
        RX packets 3594 bytes 571062 (557.6 KiB)
        RX errors 0 dropped 26 overruns 0 frame 0
        TX packets 3237 bytes 363430 (354.9 KiB)
        TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0
```

[1] Bei *IPv6* geschieht dies durch andere Mechanismen wie *Neighbor Discovery Protocol (NDP)* und *Stateless address autoconfiguration (SLAAC)*

Mithilfe dieser Tools kann in Erfahrung gebracht werden, wie die Adapter konfiguriert sind. Im Folgenden analysieren wir näher:

- *MAC*-Adresse der Netzwerkkarte, s. 2.2.1
- Aktuelle IP-Adresse des Systems, s. 2.2.2
- Subnetzmaske, s. 2.2.3
- Besteht eine aktive Verbindung mit dem Netzwerk (also Kabel mit dem Switch verbunden)? 2.2.4
- Qualität der Verbindung? (Anzahl fehlerhafter Pakete) 2.2.5
- Übertragene Datenmenge? 2.2.5

2.2.1. MAC-Adresse

Die *Media Access Control (MAC)-Adresse* ist ein eindeutiger Identifier für den Netzwerkadapter (Netzwerkkarte/Network Interface Controller (NIC)), mit deren Hilfe innerhalb eines Netzsegmentes kommuniziert werden kann. D.h. eine direkte Kommunikation zwischen zwei Knoten auf OSI-Schicht 2.
Die *MAC*-Adresse beschreibt also ein Adressierungsschema für den *Link-Layer* die dem Institute of Electrical and Electronics Engineers (IEEE) 802 Standard folgen [D'A19].

Eine *MAC*-Adresse besteht aus 48 Bit [2], was 6 Oktetts entspricht – also pro Oktett 8 Bit. Diese 8 Bit können demnach $2^8 = 256$ Werte Annehmen. Um dies einfach zu notieren, wird auf das Hexadezimalsystem zurückgegriffen, wodurch ein Oktett via zwei Hex-Werten ausgedrückt werden kann ($16^2 = 256$). Die Hexadezimale Notation ist der Regelfall.
Die eben genannten 48 Bit werden in zwei Blöcke unterteilt:

1. Teil – der Organizationally Unique Identifier (OUI), von der IEEE vergebene Herstellerkennung
   - das letzte Bit (*lsb*) des ersten Oktetts stellt dar, ob eine Unicast- oder Multicast-Adresse vorliegt
   - das vorletzte Bit zeigt, ob die Adresse global Eindeutig oder lokal administriert wird
2. Teil – „local address type" von Hersteller selbst vergeben

Eine detaillierte Beschreibung Adressierungsschema für *MAC*-Adressen kann unter [80202, S. 19ff] nachgelesen werden.

Im Listing 2.2.1 haben wir die *MAC*-Adresse des Raspberry Pis mithilfe folgender Kommandos herausgefunden.

[2] $2^{48} = 2.81474976710^{14}$ Adressen

```
ifconfig eth0 | grep -o -E '([[:xdigit:]]{1,2}:){5}[[:xdigit:]]{1,2}'
b8:27:eb:21:a0:54
ip link show eth0 | awk '/ether/ {print $2}'
b8:27:eb:21:a0:5
```

Ein Nachschlagen der *MAC*-Adresse bestätigt, dass diese Adresse zur Raspberry Pi Foundation gehört (B8:27:EB:00:00:00 ====> B8:27:EB:FF:FF:FF) [mac19]. Zu sehen ist, dass das erste Oktett $B8_{16} = 10111000_2$ ist. Hierdurch wird ersichtlich, dass es sich um eine global eindeutige Unicast-Adresse handelt.

2.2.2. Aktuelle IP-Adresse

IP-Adressen sind Adressen für die eindeutige Zuordnung von Knoten zu Netzwerken. D.h. mithilfe eines solchen Adressschemas können Knoten nicht nur direkt miteinander kommunizieren, sondern auch über die Netzgrenzen (bspw.: LAN) hinweg.
IP-Adressen arbeiten auf der Schicht drei (Network Layer) des Open Systems Interconnection (OSI)-Modells. Genaueres zu den *IP*-Protokollen kann in [Tan02, S. 439ff] oder [KR12, S. 331ff] nachgeschlagen werden.
Das *IPv4*-Adressschema besitzt nach Kurose et. al folgende Charakteristika [KR12, S. 338ff]:

- 32 Bit – $2^{32} = 4.294.967.296$ Adressen
- Schreibweise: Dezimal, à 4 Blöcke in Punktnotation
  - $32\text{Bit} : 4 = 8\text{Bit} \rightarrow 2^8 = 256\text{Werte}$
  - Pro Block: Werte $0 - 255$
- Paketorientiert: Segmentierung in Pakete, die einen Weg vom Sender zum Empfänger finden

Um die aktuelle IP-Adresse in Erfahrung zu bringen, kann wie folgt vorgegangen werden:

```
ip addr show eth0
# bzw
ifconfig eth0
# fortgeschritten Loesung
ip addr show eth0 | awk '/inet/ {print $2}'# IPv4 & 6
ip addr show eth0 | awk '/inet6/ {print $2}'#nur IPv4
ifconfig eth0 | awk '/netmask/ {print $2}'
```

Angewandt auf einem Raspberry Pi, bekommen wir folgende Ausgabe:

```
ip addr show eth0 | awk '/inet/ {print $2}'# IPv4 & 6
10.10.10.228/24
2a02:8109:83c0:371e:54d7:4903:af2e:386f/64
fe80::67a1:ecfc:4dd8:bc0d/64
ifconfig eth0 | awk '/netmask/ {print $2}'
10.10.10.228
```

Wie zu sehen ist, bekommen wir vom *DHCP*-Server die *IPv4*-Adresse 10.10.10.228 zugewiesen. Eine *IPv6*-Adresse wird uns nicht zugewiesen, daher nehmen wir an, dass im *LAN* kein *IPv6* genutzt wird.

2.2.3. Subnetzmaske

Die Subnetzmaske im *IP*-Protokoll spezifiziert, wie die einzelnen Netzwerke abzugrenzen sind [KR12, S. 340f], [rfc85]. D.h. welcher Teil einer Adresse ist die Netzadresse und welcher Anteil ist Host-Spezifisch. Dadurch können Pakete dediziert über mehrere *LAN*s verschickt werden und es ist eindeutig, welche Teilnehmer zu welchem Netzwerk gehören. Darüber hinaus kann mithilfe der Subnetzmaske die Größe eines Netzwerkes berechnet werden. Die Subnetzmaske gibt somit an, ab welcher Stelle die Netzadresse endet und der Host-Anteil beginnt (d.h. ein Maskierungsschema).

Die Subnetzmaske kann wie folgt ausgelesen werden:

```
1 ip addr show eth0 # in CIDR notation
2 # bzw direkt unter dem Eintrag netmask:
3 ifconfig eth0
4 # in schoen:
5 ip addr show eth0 | awk '/inet/ {print $2}' # CIDR notation
6 10.10.10.228/24
7 ifconfig eth0 | awk '/netmask/ {print $4}'
8 255.255.255.0
```

In unserem Fall liegt eine /24 Maske vor. D.h. von der eigentlichen 32-Bit Adresse sind 24 Bit Netzwerkanteil und 8 Bit Host-Anteil. Daraus können wir schließen, dass das Netzwerk $2^8 = 256$ Permutationen, also 256 *IP*-Adressen besitzt. Zu beachten ist, dass von diesen 256 Adressen jedoch 254 Adresse verfügbar sind, da die Netzadresse und Broadcast-Adresse mit eingerechnet aber nicht als Host-Adresse nutzbar sind.
Die eben genannte Schreibweise wird als *Classless Inter-Domain Routing (CIDR)*-Notation bezeichnet. Alternativ kann dies auch in einer dezimalen 4-Block Schreibweise erfolgen. Wobei die Zahlen der Blöcke ebenfalls zeigen, welche Bits der Adresse zu maskieren sind. Auch wird dadurch aufzeigt, wo Netzadresse endet und Host-Anteil beginnt.
Entsprechend ist bei der uns vorliegenden Konfiguration die Netzmaske: $255.255.255.0_{10}$. Dies entspricht binär $11111111.11111111.11111111.00000000_2$. Wie zu sehen ist, sind auch hier die ersten 24 Bit für den Netzanteil gesetzt (die ersten 24 Bit) und der Host-Anteil (letzten 8 Bit) besteht aus Nullen.
Beide Notationen sind also äquivalent!

2.2.4. Aktive Netzwerkverbindung

Ob die der Adpater aktiv ist, kann mit dem Werkzeug *ip link* herausgefunden werden, s. Listing 2.2.4.

```
1 ip link show eth0 | awk '/state/ {print $9}'
2 UP
```

Wie zu sehen ist, ist unser Adapter aktiv.
Darüber hinaus kann mithilfe des Tools *ping* geprüft werden, ob Teilnehmer innerhalb oder außerhalb des Netzwerkes erreichbar sind.
Jedoch ist *Ping* nicht immer zuverlässig, da beispielsweise *acICMP* „unterdrückt“sein kann (via Firewall etc.).

Mit folgenden Kommandos kann überprüft werden, ob eine Netzwerkanbindung besteht:

```
netstat -natp
(Not all processes could be identified, non-owned process info
 will not be shown, you would have to be root to see it all.)
Active Internet connections (servers and established)
Proto Recv-Q Send-Q Local Address Foreign Address State PID/Program name
tcp 0 0 0.0.0.0:22 0.0.0.0:* LISTEN -
tcp 0 0 127.0.0.1:5432 0.0.0.0:* LISTEN -
tcp 0 164 10.10.10.228:22 10.10.10.228:57190 ESTABLISHED -
tcp6 0 0 :::22 :::* LISTEN -
tcp6 0 0 ::1:5432 :::* LISTEN -
# alternativ:
netstat -tulpn
iftop
netstat -s
# oder aequivalent zu netstat->ss
ss -s
Total: 318 (kernel 0)
TCP: 5 (estab 1, closed 0, orphaned 0, synrecv 0, timewait 0/0), ports 0

Transport Total IP IPv6
* 0 --
RAW 1 0 1
UDP 7 3 4
TCP 5 3 2
INET 13 6 7
FRAG 0 0 0
```

Ping

Wenn ein Rechner mit seinem Hostname angepingt werden soll, dann eignet sind sich folgendes Vorgehen:

```
# Sende 3 Pakete nacheinander an Adresse
ping -c3 www.htw-berlin.de
# sehr schoener hostname!
PING moehre.htw-berlin.de (141.45.7.250) 56(84) bytes of data.
64 bytes from moehre.htw-berlin.de (141.45.7.250): icmp_seq=1 ttl=245 time=15.4 ms
64 bytes from moehre.htw-berlin.de (141.45.7.250): icmp_seq=2 ttl=245 time=11.8 ms
64 bytes from moehre.htw-berlin.de (141.45.7.250): icmp_seq=3 ttl=245 time=11.1 ms

```

```
---moehre.htw-berlin.de ping statistics ---
3 packets transmitted, 3 received, 0% packet loss, time 2002ms
rtt min/avg/max/mdev = 11.159/12.830/15.440/1.871 ms
```

Entsprechend sieht das Vorgehen für das Anpingen von *IPv4*-Adressen aus (*IPv6* ist dazu äquivalent zu nutzen).

```
#zu c)
ping -c3 8.8.8.8 # google dns oder
ping -c3 141.45.146.48 # uranus server der htw
# Laesst kein ping/ICMP zu
ping -c3 -p 22 uranus.f4.htw-berlin.de
PATTERN: 0x22
PING uranus.f4.htw-berlin.de (141.45.146.48) 56(84) bytes of data.

---uranus.f4.htw-berlin.de ping statistics ---
3 packets transmitted, 0 received, 100% packet loss, time 2077ms
```

Um den Laborrouter zu erreichen muss die Adresse 10.10.10.254 angepingt werden.

```
#zu d)
ping -c3 10.10.10.254
```

Um den lokalen Netzwerkstack zu testen, kann wie folgt vorgegangen werden:

```
#zu e)
ping -c3 127.0.0.1 # bzw.
ping -c3 localhost
PING localhost(localhost (::1)) 56 data bytes
64 bytes from localhost (::1): icmp_seq=1 ttl=64 time=0.026 ms
64 bytes from localhost (::1): icmp_seq=2 ttl=64 time=0.027 ms
64 bytes from localhost (::1): icmp_seq=3 ttl=64 time=0.026 ms

---localhost ping statistics ---
3 packets transmitted, 3 received, 0% packet loss, time 2026ms
rtt min/avg/max/mdev = 0.026/0.026/0.027/0.004 ms
```

2.2.5. Fehlerhafte Pakte & übertragene Datenmenge

Fehlerhafte Pakete können mithilfe folgender Kommandos ermittelt werden:

```
#einfache Loesung -direkt unter dem Eintrag error & dropped:
ifconfig eth0
#fuer fortgeschrittene
ifconfig eth0 | awk '/errors/ {print $3}'
ifconfig eth0 | awk '/dropped/ {print $3}'
ifconfig | grep -E "^\w|errors.* " | sed 's/pack.*errors:/Errors:/g'
| sed 's/ drop.*//g' | sed 's/HW.*//g'
#Achtung Ein-Zeiler die Single-Quotes muessen angepasst werden
```

Wobei der Einzeiler am direktesten anzeigt, wie viele fehlerhafte Pakete vorliegen.

```
ifconfig eth0 | grep -E "^\w|errors.* " | sed 's/pack.*errors:/Errors:/g' | sed 's/ drop.*//g' | sed 's/HW.*//g'
eth0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST> mtu 1500
        RX errors 0
        TX errors
```

Mit folgenden Kommandos kann ermittelt werden, wie viele Daten übertragen wurden:

```
#einfache Loesung -direkt unter dem Eintrag packets & bytes:
ifconfig eth0
#fuer fortgeschrittene
ifconfig eth0 | awk '/bytes/ {print $0}'
nload
cat /proc/net/dev
netstat -i
cat /sys/class/net/eth0/statistics/rx_bytes
```

DHPC Off Als letzte Aufgabe soll der *DHCP*-Client-Daemon permanent ausgeschaltet werden, da im Folgeabschnitt ein statisches Netzwerk ohne *DHCP* aufgesetzt wird. Der Name deutet die Funktionalität des Dienstes bereits an: es handelt sich um einen Client-Daemon, also ein Programm, das von einem Server Dienste in Anspruch nimmt. Entsprechend muss es einen *DHCP*-Server geben. Dieser ist im Labor unter der Adresse 10.10.10.254 erreichbar.
Das ausschalten des *dhcpcd* erfolgt durch die Kommandos:

```
1 sudo systemctl status dhcpcd
2 sudo systemctl stop dhcpcd
3 sudo systemctl disable dhcpcd
```

Wobei Zeile 1 des Listing 2.2.5 den Status des Daemons wiedergibt, Zeile 2 den Dienst stoppt und Zeile 3 den Dienst permanent abschaltet – d.h. aus dem Initsystem ausschließt.

Der Networking-Service soll aus bequemlichkeitsgründen permanent eingeschaltet werden. Der *networking.service* ist eigentlich kein richtiger Service, sondern sorgt lediglich dafür, dass die NICs automatisch hoch- und runter gefahren werden.
Mit folgendem Befehl kann dies umgesetzt werden:

```
1 sudo systemctl status networking.service
2 sudo systemctl start networking.service
3 sudo systemctl enable networking.service
```

2.3. Switched LAN

Der zweite Aufgabenteil bestand darin ein eigenes geswitches *LAN* aufzusetzen. Dieses Netzwerk soll aus den Überlegungen der Hausaufgaben hervorgehen.
Die Planung eines Netzwerkes basiert oft auf Topologien, also den formellen Beschreibungen des Netzwerkes. Dadurch wird klar, wie die Anordnung der Knoten (Rechner etc.)

und Kanten (Verbindungen zwischen den Knoten) aussehen kann. Ebenfalls wird an dieser Stelle klar, ob in der Planungsphase bereits erste Fehler auftauchen. Beispielsweise ob es Redundanzen gibt, die mögliche Ausfälle abfangen oder wie die Lastverteilung im Netzwerk aussieht.
Mithilfe solcher Topologien ist es ebenfalls möglich mathematisch zu beweisen, wie die Eigenschaften eines Netzwerkes sind!

Für unser Netzwerk haben wir folgende Topologie gewählt:

- physikalisch (Layer 1): Multibus – Switch verbindet die Endknoten via Bus miteinander
- physikalisch (Layer 2): Stern-Topologie – Switch als zentraler Knoten, der Frames weitergibt
- Layer 3: Switch ist transparent und auf, es kommt also darauf an...

Entsprechend zeigt die Abbildung 1 die von uns genutzt Topologie.

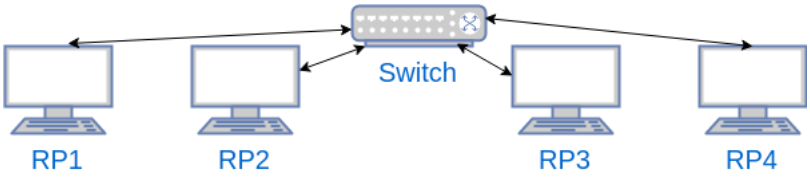

Abbildung 1: Skizze des LANs aus topologischer Sicht mit vier Raspberry Pis und einem Switch.

Darüber hinaus zeigt Abbildung 2 die von gewählten *IPv4*-Adressen samt Subnetzmaske und *MAC*-Adresse. Zu beachten ist, dass der Switch selbst keine IP-Adresse kennt, da dieser ein Layer 2 Gerät ist und nur *MAC*-Adressen kennt.
Jedoch soll dieser Switch symbolisieren, wie das Netzwerk aussieht, indem er die Neztadresse „trägt“.

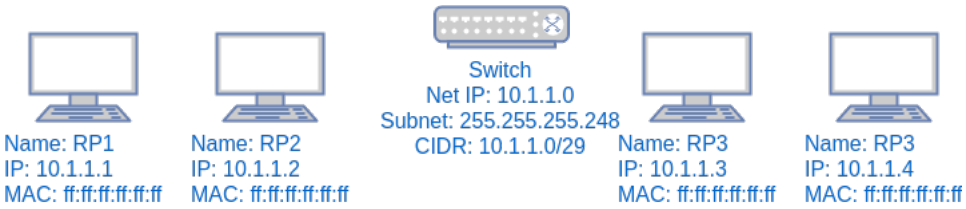

Abbildung 2: Skizze des LANs nach dem gegebenen Adressierungsschemas.

Da wir pro Bankreihe vier Raspberry Pis miteinander vernetzen sollen, muss die Subnetzmaske /29 bzw. 255.255.255.248 lauten. Wodurch wir die ersten 29 Bits für die Netzmaske deklarieren und die letzten 3 Bits für den Host-Anteil. Mit diesen 3 Bits können

$2^3 = 8$ Rechner adressiert werden. Wir benötigen vier Adressen für die Raspberry Pis und zwei Standardadressen. Die erste Adresse ist die Netzadresse, die letzte Adresse ist die Broadcastadresse. Jedoch ergibt dies eine interne Fragmentierung von zwei Adressen,d.h. diese Adressen sind allokiert werden aber nicht genutzt.
Mit diesem Vorgehen ist das Labor wie folgt strukturiert (s. Tabelle 1):

Tabelle 1: Beispielkonfiguration

Gruppe	**Netzwerk IP**	**Subnetzmaske**	**CIDR**
Fractals	10.1.1.0	255.255.255.248	10.1.1.0/29
1337	10.1.2.0	255.255.255.248	10.1.2.0/29
...			

Jede Bankreihe hat eine feste IP-Range, die sie nutzen kann. Eine Kommunikation der Netzwerke über diese Segmente hinaus ist nicht möglich.

2.3.1. On the fly Konfiguration

Die Konfiguration der Raspberry Pis wird im Folgenden beispielhaft für ein Gerät gezeigt. Bei allen Kommandos muss das Schlüsselwort *sudo* vor dem eigentlichen Befehl angeführt werden, da nicht jeder Nutzer die Netzwerkadapter konfigurieren können soll. Das Listing 2.3.1 zeigt, wie eine *IPv4*-Adresse gesetzt und aktiviert wird.

```
# via iproute2
ip addr add 10.1.1.1/29 dev eth0
ip link set eth0 up
# via net tools
ifconfig eth0 10.1.1.1 netmask 255.255.255.248 up
```

Mithilfe der Befehle *ip addr show eth0* oder *ifconfig eth0* kann anschließend überprüft werden, ob der Adapter die gewünschte Konfiguration hat.
Um zu testen, ob die Raspberry Pis sich tatsächlich erreichen können, kann das Werkzeug *ping* genutzt werden. Die entsprechende Nutzung kann im Listing 2.2.4 nachverfolgt werden.

2.3.2. Persistente Umsetzung

Da diese Umsetzung lediglich temporärer Natur ist, wird nach einem Neustart des Systems jegliche Konfiguration verworfen. Entsprechend der Aufgabenstellung soll eine persistente Lösung umgesetzt werden. Das Listing 2.3.2 zeigt diese.

```
# /etc/network/interfaces
auto lo # automatische setzen des loopback devices (localhost)
iface lo inet loopback # setze loopback ipv4 adresse

auto eth0 # hochfahren des interfaces zur boot zeit
allow-hotplug eth0 # erlaubt kabel wechsel waehrend des betriebs
```

```
7 iface eth0 inet static # statische ip addr fuer interface eth0
8         address 10.1.1.1 # ip addr -statisch
9         netmask 255.255.255.248 # subnetzmaske
```

Wie zu sehen ist, erfolgt die Konfiguration nicht via Kommandozeilentools, sondern in einer Datei. Diese Datei befindet sich bei *Raspbian* standardmäßig im Verzeichnisbaum unter `/etc/network/interfaces`. Die `interfaces`-Datei darf ebenfalls nur mit *root*-Rechten beschrieben werden.
Zeile 2 und 3 befassen sich ausschließlich mit der Konfiguration des Loopback-Devices. Diese ist für die Übungsaufgabe nicht zwingend notwendig. Die Zeilen 5-9 zeigen die Konfiguration des NIC *eth0*. Zeile 5 zeigt, dass der Adapter automatisch hochgefahren werden soll. Zeile 6 kümmert sich um das Hot-Plugging. Die Zeile 7 zeigt, dass es sich um eine statische *IPv4*-Adressierung des *eth0*-Adapters handelt – Stichwort *inet*. Zeile 8 ist die Notation für die Adresszuweisung und Zeile 9 gibt die zu nutzende Netzwerkmaske an. Mit dieser Konfiguration kann erneut via *ping* getestet werden, ob der Netzwerkstack korrekt eingerichtet wurde.

Anmerkung: In diesem Beispielprotokoll fehlen ein paar schicke Screenshots, die zeigen, dass *ping* oder *ifconfig* etc. korrekt funktionieren. Ich würde es sehr begrüßen, solche Screenshots zu sehen.

3. Schluss

Hier könnten Verweise auf Alternativen, Ausblicke und Gedanken festgehalten werden. Da ich diese nicht habe, erspare ich dies dem Leser.

A. Abkürzungsverzeichnis

CIDR Classless Inter-Domain Routing 8
DHCP Dynamic Host Configuration Protocol 3
GNU GNU is Not Unix
IEEE Institute of Electrical and Electronics Engineers 6
LAN Local Area Network 3
MAC Media Access Control 6
NDP Neighbor Discovery Protocol 5
NIC Network Interface Controller 6
OUI Organizationally Unique Identifier 6
OSI Open Systems Interconnection 7
PID Process Identifier 4
SoC System-on-a-Chip 3
SLAAC Stateless address autoconfiguration 5

Abbildungsverzeichnis

B. Quellen

Literatur

[80202] Ieee standard for local and metropolitan area networks: Overview and architecture. *IEEE Std 802-2001 (Revision of IEEE Std 802-1990)*, pages 1–48, Feb 2002.

[D'A19] John D'Ambrosia. Ieee 802 lan/man standards committee. http://www.ieee802.org/, 2019. Letzter Aufruf: 26.04.2019.

[Fou19] Raspberry Pi Foundation. Raspberry pi 3 model b+. https://www.raspberrypi.org/products/raspberry-pi-3-model-b-plus/, 2019. Letzter Aufruf: 26.04.2019.

[KR12] James F. Kurose and Keith W. Ross. *Computer Networking: A Top-Down Approach (6th Edition)*. Pearson, 6th edition, 2012.

[mac19] Mac address lookup. https://www.macvendorlookup.com/, 2019. Letzter Aufruf: 26.04.2019.

[rfc85] Internet Standard Subnetting Procedure. RFC 950, August 1985.

[Tan02] Andrew Tanenbaum. *Computer Networks*. Prentice Hall Professional Technical Reference, 4th edition, 2002.

[Wik19a] Arch Wiki. Systemd. https://wiki.archlinux.org/index.php/Systemd, 2019. Letzter Aufruf: 26.04.2019.

[Wik19b] Debian Wiki. Systemd. https://wiki.debian.org/Debate/initsystem/systemd, 2019. Letzter Aufruf: 26.04.2019.