INSTITUT FÜR
INFORMATIK
DER LUDWIG-MAXIMILIANS-UNIVERSITÄT MÜNCHEN

LS Prof. Kranzlmüller

Praktikum Rechnernetze

Kapitel 1: Erste Schritte

MNM
TEAM
Munich Network Management Team

1 Erste Schritte

Inhaltsverzeichnis

1.1 Ablauf des Praktikums 4
1.1.1 Präsenztermine 4
1.1.2 Ausarbeitungen 4
1.2 Adressierung und Wegewahl 5
1.2.1 Vermittlung und Wegewahl 5
1.2.2 Das IP-Protokoll (Version 4) 6
1.3 RNP Infrastruktur, Linux und Tools 10
1.3.1 Linux, eine knappe Einführung 10
1.3.2 Arbeiten mit der virtuellen Infrastruktur 11
1.3.3 IProute2 – IP-Konfiguration 12
1.3.4 tcpdump – Datenanalyse 13
1.4 Aufgaben 14

Willkommen im Praktikum Rechnernetze! Das Praktikum bietet Einblicke in die technischen Details rund um Rechnernetze und Netzmanagement. Das Praktikum ist in fünf Blöcke untergliedert.

In diesem ersten Block werden Sie lernen mit Linux und dessen grundlegenden IP-Funktionen umzugehen, da der TCP/IP-Stack die Basis aller Applikationen ist, mit denen wir Daten über Netze versenden können. Des weiteren erläutert die Einführung den Umgang mit der Praktikums-Infrastruktur und Anforderungen an die Ausarbeitungen, die Sie im Rahmen dieses Praktikums anfertigen werden.

Im Anschluss an diese Einführung werden Sie im zweiten Block Erfahrungen im Umgang mit virtualisierten Komponenten und deren Konfiguration sammeln. Im Vordergrund steht hier die ISO-OSI Schicht 2.

Während der Bearbeitung des dritten Blocks werden Sie sich mit den erweiterten Fragestellungen rund rum OSI Schicht 3 beschäftigen. Dabei werden Sie unterschiedliche Aspekte der Protokolle der IP-Familie untersuchen und eine Routinghierarchie erstellen, die autonomen Systemen nachempfunden ist.

Darauf aufbauend lernen Sie im vierten Block Konzepte des Software-Defined-Networkings (SDN) kennen, welche Ihnen ermöglichen programmatisch Einfluss auf Ihr Netz zu nehmen.

Im letzten Block werden Sie ein Projekt zum Thema Rechnernetze praktisch implementieren. Eigene Vorschläge können hier auch umgesetzt werden.

1.1 Ablauf des Praktikums

Das Praktikum Rechnernetze (RNP) ist unterteilt in fünf Praktikumsblöcke: Einführung, virtualisierte Netze, Autonome Systeme, Software-Defined Networking (SDN) und Projektarbeit. Jeder Block enthält Aufgaben, die von den Teilnehmern bearbeitet werden.

Die Bearbeitung eines Blocks wird durch die Anfertigung einer Ausarbeitung dokumentiert. Die Arbeitszeit kann frei gewählt werden, jedoch müssen alle Ausarbeitungen spätestens am vorgegebenen zeitlichen Ende eines Praktikumsblocks abgegeben werden.

1.1.1 Präsenztermine

Jeweils zum Start eines Aufgabenblocks findet in der **Oettingenstraße 67** eine Praktikumsbesprechung statt, in der Wissen vertieft und Probleme besprochen werden. Zusätzlich zu diesen Treffen (mit Anwesenheitspflicht) sollte jede Gruppe mindestens einmal die Woche zusammen an den Aufgaben arbeiten.

Mit dem Tutor wird ein fester Termin abgestimmt, so dass regelmäßig die Möglichkeit besteht bei der Bearbeitung der Aufgaben Fragen zu stellen.

1.1.2 Ausarbeitungen

Die Ausarbeitungen dokumentieren die Bearbeitung von Aufgabenblättern. Im wesentlichen besteht eine Antwort aus einer textuellen Zusammenfassung des durchgeführten Versuchs, einem Netzplan, der den Aufbau des Versuchs und alle beteiligten Komponenten zeigt, sowie alle abgesetzten relevanten Befehlen auf den Systemen und die entsprechende Ausgabe, die diese Befehle erzeugt haben.

Abgesehen von einem strukturierten Aufbau müssen Ausarbeitungen auch einer äußeren Form genügen. Deshalb steht für das Verfassen von Ausarbeitungen ein LaTeX-Rahmen zur Verfügung. Teil des Rahmens ist ein Makefile, dass durch den Aufruf von `make` ein PDF generiert. Für den schnellen Einstieg in LaTeX enthält der Rahmen ein Beispieldokument an dem Sie sich orientieren können. Übersetzt man das Beispieldokument (durch entpacken des Archivs und Aufruf von `make`) so erhält man eine kurze Anleitung für die häufigsten Befehle.

Ein großer Vorteil des Rahmens ist die breite Unterstützung für verschiedene Grafikformate, wodurch der Aufwand zum Exportieren und Konvertieren von Bildern deutlich minimiert wird. Für ein optimales Ergebnis sollten Sie ausschließlich Vektorgrafiken verwenden. Unterstützte Formate für Vektorgrafiken sind EPS, PDF, Xfig, Dia, und SVG. Andere Programme wie z.B. OpenOffice verfügen über Funktionen um Dateien in das PDF-Format zu exportieren.

Die Netzpläne dienen der Dokumentation als Beschreibung der Topologie und der beteiligten Komponenten. Da nicht stets alle zur Verfügung stehenden Komponenten eingesetzt werden, bilden Netzpläne die Grundlage der Beschreibung eines Versuchsaufbaus.

Ein Netzplan muss alle (relevanten) Komponenten, Verbindungen und Bezeichnungen enthalten.

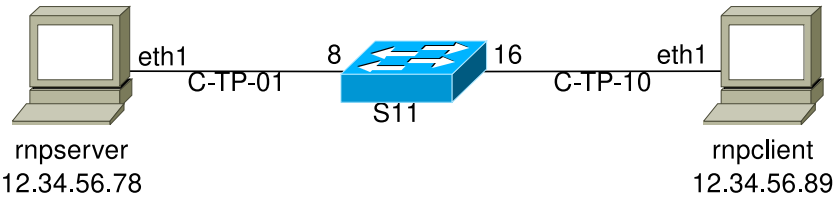


Abb. 1.1: Ein Netzplan, der zwei Rechner zeigt, die über einen Switch miteinander verbunden sind

Abbildung 1.1 zeigt ein Beispiel eines Netzplans für einen Aufbau in dem zwei Rechner über einen Switch miteinander verbunden sind. Der Netzplan dokumentiert genau welche Komponenten verwendet wurden und wie diese kombiniert wurden. In diesem Beispiel ist die Schnittstelle eth1 des Rechners `rnpserver` über Kabel C-TP-01 an Port 8 des Switches S11 angeschlossen. Analog ist die Schnittstelle eth1 des Rechners `rnpclient` über Kabel C-TP-10 an Port 16 des Switches S11 angeschlossen. Die Rechner haben die IPv4-Adressen 12.34.56.78 bzw. 12.34.56.89.

1.2 Adressierung und Wegewahl anhand von IPv4

Aufbauend auf der Sicherungsschicht, die dafür zuständig ist Rahmen durch ein LAN zu transportieren, werden mit den Funktionen der Vermittlungsschicht (Schicht 3 des ISO-OSI Referenzmodells) Nachrichten von der Quelle bis zum endgültigen Ziel übertragen. Dabei können Teilnetze durchquert werden, die unterschiedliche Schicht 2 Implementierungen einsetzen. Da die Vermittlungsschicht Datagramme bis zum endgültigen Ziel überträgt, müssen Endpunkte über alle Teilnetze hinweg eindeutig adressierbar sein.

1.2.1 Vermittlung und Wegewahl

Innerhalb einer Broadcast-Domäne (z.B. Ethernet-Bus) können Nachrichten direkt an einen Empfänger zugestellt werden: alle Kommunikationsteilnehmer „hören" die Nachricht auf dem Broadcastmedium. Der Adressat erkennt an der Zieladresse (es ist seine eigene!), dass er die Nachricht auszuwerten hat. Soll eine Nachricht nach außerhalb der Broadcast-Domäne des Senders übertragen werden, sind Koppelkomponenten erforderlich, die eine *Vermittlung* der Nachricht ermöglichen. Sie kennen bereits die Vermittlung auf der Sicherungsschicht, in geswitchten Ethernet-LANs: ein Switch entscheidet anhand einer ihm bekannten Ziel-MAC-Adresse eines Rahmens, an welche(n) seiner Ports der Rahmen ausgegeben werden soll. Kennt der Switch die Zieladresse des Rahmens noch nicht, so gibt er den Rahmen auf allen Ports aus.

Zwischen LANs übernimmt die Schicht 3 des ISO-OSI Modells, die Vermittlungsschicht (engl. network layer), die Weitergabe von Nachrichten. Die Komponenten der Vermittlungsschicht heißen *Router*. Ähnlich wie Switches verfügen sie über eine Anzahl Schnittstellen, die in verschiedene Subnetze führen. Ein Router entscheidet anhand einer *Routing-Tabelle* und der Zieladresse des Schicht 3 Protokolls, in welches Teilnetz eine Nachricht vermittelt werden soll.

Abbildung 1.2 zeigt drei Ethernet-LANs, zwei Busse und ein geswitchtes LAN, die mittels eines Routers verbunden sind.

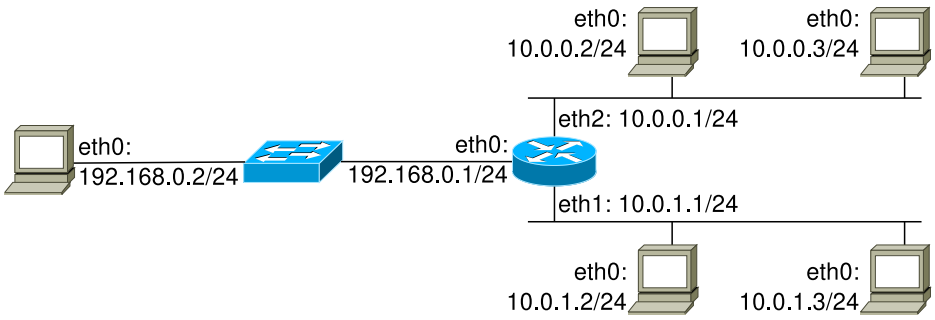


Abb. 1.2: Drei Ethernet-LANs, verbunden durch einen Router

1.2.2 Das IP-Protokoll (Version 4)

Die am weitesten verbreitete Schicht 3 Implementierung ist das IP-Protokoll in der Version 4 (IPv4, [RFC 791]). IPv4 ist ein verbindungsloses Schicht 3 Protokoll, d.h. alle Datagramme werden *unabhängig* voneinander zugestellt. Der Dienst, den IPv4 höheren Schichten zur Verfügung stellt, ist lediglich die Zustellung von Datagrammen zu einem bestimmten Endpunkt. IPv4 stellt nicht sicher, dass Datagramme in der selben Reihenfolge empfangen werden, in der sie gesendet wurden. Ebenso sieht IPv4 keine Funktionen vor, die sicherstellen, dass die Daten unverfälscht am Ziel ankommen.

Seine Nachfolgeversion IPv6 ([**?**]) ist nicht weniger relevant, auch wenn es aktuell noch nicht in gleichem Maße verbreitet ist.

Adressierung

Für die Adressierung in einem IPv4-Netz werden hierarchische 32-Bit Nummern verwendet. Das heißt *IPv4-Adressen* (kurz: IP-Adressen) dienen sowohl der Identifizierung der Gegenstellen, als auch der Strukturierung des Adressraums. IP-Adressen setzen sich aus einer Netz-ID und einer Host-ID zusammen. IP-Adressen mit der selben Netz-ID gehören zum selben *Subnetz*. Ursprünglich wurden IP-Adressen in Subnetzklassen fester Größe eingeteilt (vgl. Vorlesung RNVS), wodurch die Anzahl von IP-Adressen in einem Subnetz immer 256, 65536 oder 16777216 betrug. In heutigen Implementierungen ist die

Länge der Netz-ID variabel, um Subnetze unterschiedlicher Größe anlegen zu können. Dadurch kann bei der Vergabe von Adressblöcken die Anzahl der IP-Adressen besser an die tatsächlich benötigte Menge angepasst werden, wodurch weniger „Verschnitt" (ungenutzte IP-Adressen) entsteht. Dieser Ansatz ist als Classless Inter-Domain Routing (CIDR) in [RFC 1519] spezifiziert.

In jedem Subnetz sind zwei Adressen reserviert: die Netzadresse und die Broadcast-Adresse. Die Netzadresse ist die IP-Adresse, bei der alle Bits der Host-ID 0 sind. Die Netzadresse ist die Adresse mit dem numerisch kleinsten Wert im Subnetz. Im Gegensatz dazu sind die Bits der Host-ID in der Broadcast-Adresse alle 1. Somit ist die Broadcast-Adresse die IP-Adresse mit dem numerisch höchsten Wert im Subnetz.

Routing-Tabellen

Die Einträge einer Routing-Tabelle geben anhand von Zieladressen Routing-Entscheidungen vor. Aufgrund der hierarchischen Vergabe von IP-Adressen muss eine Routing-Tabelle nicht einen Eintrag pro IP-Adresse enthalten, sondern einen Eintrag pro Subnetz. Somit benötigt der Router aus Abbildung 1.2 drei Einträge in seiner Routing-Tabelle, um Nachrichten an alle fünf dargestellten Rechner weiterleiten zu können:

Ziel	next-hop
10.0.0.0/24	10.0.0.1
10.0.1.0/24	10.0.1.1
192.168.0.0/24	192.168.0.1

Ein Eintrag besteht aus einem Ziel-Subnetz und der IP-Adresse der Komponente, die das Ziel erreichen kann (next-hop). Da der Router über eine direkte Verbindung in jedes der drei Subnetze verfügt, sind alle next-hop-Einträge auf der rechten Seite der Routing-Tabelle die eigenen IP-Adressen des Routers. Der Router kann in diesem Fall jedes Ziel *direkt*, d.h. ohne Zwischenschritte (engl. hops), erreichen.

Die Rechner in Abbildung 1.2 sind mit genau einem Subnetz direkt verbunden. Sendet einer davon eine Nachricht, muss der Router diese Nachricht zwischen Sender und Empfänger vermitteln. Sollen Nachrichten zwischen 10.0.0.3 und 10.0.1.3 ausgetauscht werden, so benötigen die Rechner (mindestens) Routing-Tabellen wie in Tabelle 1.1.

In manchen Situationen ist es nicht sinnvoll oder möglich für jedes erreichbare Subnetz einen eigenen Eintrag in der Routing-Tabelle anzulegen. Man kann z.B. auf dem Rechner zuhause kaum für jedes über das Internet erreichbare Subnetz einen Eintrag anlegen. Für diesen Fall legt man einen Standard-Eintrag in der Routing-Tabelle an, der genau dann ausgelesen wird, wenn kein anderer Eintrag für eine IP-Adresse gefunden werden kann. Dieser Eintrag wird häufig *default-Route* genannt und enthält als next-hop das *Standard-Gateway* (engl. default gateway). Umgekehrt ist es unter Umständen erwünscht Nachrichten an einen bestimmten Rechner über einen speziellen Pfad zu schicken. Solche Einträge in der Routing-Tabelle, die für genau einen Rechner gelten, heißen *Host-Routen*. Host-Routen können als Subnetz mit genau einer IP-Adresse gesehen werden, also als ein Subnetz mit 32 festen Bits (/32).

auf Rechner 10.0.0.3:			Erläuterungen:
Ziel	next-hop		
10.0.0.0/24	10.0.0.3	←	direkte Route in das angeschlossene Subnetz
10.0.1.0/24	10.0.0.1	←	Route in das andere Subnetz über den Router
auf Rechner 10.0.1.3:			Erläuterungen:
Ziel	next-hop		
10.0.1.0/24	10.0.1.3	←	direkte Route in das angeschlossene Subnetz
10.0.0.0/24	10.0.1.1	←	Route in das andere Subnetz über den Router

Tab. 1.1: Routing-Tabellen der Rechner mit den IP-Adressen 10.0.0.3 und 10.0.1.3 mit Erläuterungen

In der Praxis liefern verschiedene Programme leicht unterschiedliche Formate für ein und dieselbe Routing-Tabelle, wie etwa in Abbildungen 1.3, 1.4 und 1.5. Routing-Tabellen gängiger IP-Implementierungen beinhalten meist mehr Informationen als ein Ziel und den dazugehörigen next-hop. Eine weitere oft gespeicherte Information ist die Schnittstelle, über die ein Ziel erreicht werden kann. Manchmal wird bei direkt erreichbaren Zielen 0.0.0.0 als next-hop, statt einer eigenen IP-Adresse gespeichert.

```
Kernel IP routing table
Destination Gateway         Genmask       Flags MSS irtt Iface
localnet    *               255.255.255.0 U       0    0 eth1
default     10.153.211.254 0.0.0.0       UG      0    0 eth1
```

Abb. 1.3: Beispiel 1, erzeugt mit `netstat`

```
Kernel IP routing table
Destination  Gateway         Genmask       Flags Metric Ref Use Iface
10.153.211.0 0.0.0.0         255.255.255.0 U          0   0   0 eth1
0.0.0.0      10.153.211.254 0.0.0.0       UG         0   0   0 eth1
```

Abb. 1.4: Beispiel 2, erzeugt mit `route`

```
10.153.211.0/24 dev eth1 proto kernel scope link src 10.153.211.1
default via 10.153.211.254 dev eth1
```

Abb. 1.5: Beispiel 3, erzeugt mit `ip`

Wegewahl

Bei der Einteilung von IP-Adressen in Subnetzklassen ist der Algorithmus zur Auswahl einer Route vergleichsweise einfach (vgl. Folien RNVS): die ersten (bis zu vier) Bits einer IP-Adresse bestimmen die Subnetzklasse und damit die Länge der Netz-ID. Die Netz-ID wird ausgelesen und der nächste Zwischenschritt (next hop) in der Routing-Tabelle nachgeschlagen.

Mit der Einführung von CIDR ist der Entscheidungsprozess komplexer geworden. Die beiden wichtigsten Neuerungen diesbezüglich sind, dass Subnetze nicht mehr zu einer

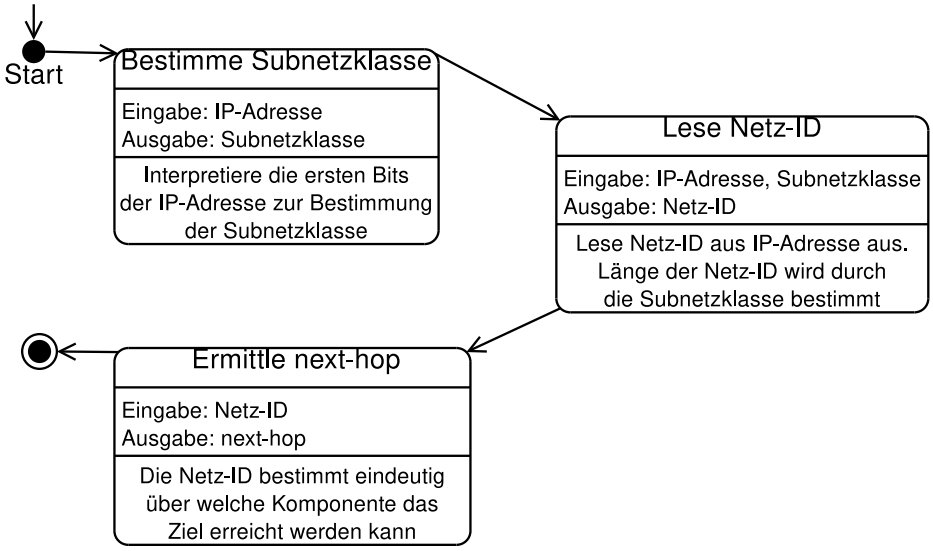


Abb. 1.6: Schematische Darstellung eines Automaten zur Wegewahl

bestimmten Klasse gehören und die Länge der Netz-ID variabel ist. Die Folge dieser Veränderungen ist, dass die Länge der Netz-ID nicht mehr an der IP-Adresse selbst abgelesen werden kann, sondern durch die Einträge in der Routing-Tabelle gegeben werden muss. Anstatt einmal die Netz-ID auszulesen und einen passenden Eintrag dafür zu suchen muss nun die variable Länge berücksichtigt werden.

Longest prefix match Empfängt ein Router ein IP-Paket, das an die Ziel-IP-Adresse 10.0.0.3 adressiert ist, so liest der Router die Ziel-IP-Adresse aus und durchsucht die Routing-Tabelle nach dem präzisesten zutreffenden Eintrag.

Beispiel Angenommen die Routing-Tabelle des Routers enthält die beiden Einträge:

Ziel	next-hop
10.0.0.0/20	10.0.10.1
10.0.0.0/24	10.0.20.1

Beide Einträge können für die Ziel-IP-Adresse 10.0.0.3 angewendet werden: die ersten 20 Bit der IP-Adresse 10.0.0.3 passen auf den ersten Eintrag, ebenso wie die ersten 24 Bit auf den zweiten Eintrag passen. In diesem Fall wird der präzisere Eintrag, mit der längeren Netz-ID, ausgewählt. Diese Stategie heißt *longest prefix match*.

1.3 RNP Infrastruktur, Linux und Tools

Die Infrastruktur, die für die Durchführung des RNP zur Verfügung steht setzt sich zusammen aus virtuellen Maschinen. Der Zugang zu den virtuellen Maschinen (VM) erfolgt über gruppeXY.rnp.lab.nm.ifi.lmu.de. Diese sind aus dem MWN, zum Beispiel den cip-Pools, erreichbar. Die VMs selbst haben standardmäßig keinen Zugang zum MWN oder dem Internet. Jeder Praktikumsgruppe stehen sieben VMs zur Verfügung, mit denen die Versuche ab OSI Schicht 3 und höher durchgeführt werden. Der Einsatz von virtuellen Maschinen erlaubt es größere Netze mit mehr Knoten zu betreiben, ohne den Management-Aufwand und Platzbedarf vieler physischer Systeme bewältigen zu müssen.

1.3.1 Linux, eine knappe Einführung

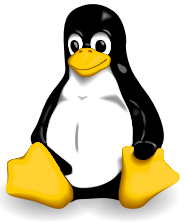


Abb. 1.7: Tux

Auf den VMs an denen Sie die Versuche durchführen werden, ist OpenWrt und Debian (GNU/Linux) installiert. Im Rahmen dieses Praktikums werden Sie Linux meistens über ein Terminal bedienen, auf dem ein Kommandozeileninterpreter (Shell) läuft. Nach dem erfolgreichen Anmelden am System startet Linux automatisch eine Shell. Die Standardeinstellung hierfür ist die Bourne-Again SHell (Bash).

Jeder Benutzer hat ein eigenes Verzeichnis (Home-Verzeichnis). Für den Benutzer `root` ist dies das Verzeichnis `/root`. Für alle anderen Benutzer ist das Home-Verzeichnis ein Unterverzeichnis von /home, das dem Benutzernamen entspricht (/home/<Benutzername>). Wann immer sich ein Benutzer in seinem Home-Verzeichnis befindet, wird dieser Pfad von der Bash mit ~ (Tilde) abgekürzt. Der Aufbau der Eingabezeile kann verändert werden, wodurch der hier beschriebene Aufbau nicht allgemein für alle Benutzer und Systeme gilt. Es hat sich jedoch durchgesetzt, dass am Ende einer Eingabezeile des Benutzers `root` das #-Symbol steht.

Der wichtigste Befehl beim Arbeiten mit der Bash ist **`man`**. Bis auf wenige Ausnahmen existiert für jedes Programm eine Anleitung. Der Befehl `man` dient dem Lesen von Anleitungen. Ein Befehl auf der Kommandozeile ist häufig ein Programmaufruf, wie z.B. `pwd`, `ls` und `man`. Um die Anleitung für ein bestimmtes Programm einzusehen geben Sie `man <Programm>` ein. So erhalten Sie z.B. die Anleitung zu `man` selbst mittels:

```
# man man
```

und die Anleitung zu `pwd` durch:

```
# man pwd
```

Beachten Sie, dass OpenWrt optimiert ist für Router mit wenig Speicherplatz. Aus diesem Grund steht ihnen der Befehl `man` auf OpenWrt (router1-router3) nicht zur Verfügung. Sie können stattdessen auf die äußere Managementmaschine ausweichen. Die Verwendung der Befehle zwischen Debian und OpenWrt ist meistens identisch.

Abweichungen davon, wie zum Beispiel die Dokumentation von Konfigurationsdateien, finden Sie in der Online-Dokumentation[1] zu OpenWrt.

Das Bash-Tutorial [Garr 08] ist ein guter Ausgangspunkt für die Einarbeitung. Insbesondere sei auf die Abschnitte des Tutorials hingewiesen, die sich mit "Input-/Output-redirection" und Bash-Programmierung beschäftigen. Mit "Output-redirection" kann man Programmausgaben in eine Datei umlenken und anschließend in Ausarbeitungen einfügen. Bash-Programmierung ermöglicht es kleine Skripte zu schreiben, die oft durchgeführte Aufgaben vereinfachen.

1.3.2 Arbeiten mit der virtuellen Infrastruktur

Jeder Gruppe steht eine virtuelle Infrastruktur in Form von sieben VMs zur Verfügung; drei PCs und vier Router. Alle sieben VMs sind Linux-Rechner. Der Unterschied zwischen den VM-Arten besteht darin, dass die PCs nur eine Verbindung in die virtuelle Infrastruktur haben, während die Router mit mehreren Maschinen verbunden sind. Die Rechnernamen der VMs sind `pc1`, `pc2`, `pc3`, `router1`, `router2`, `router3` und `router4`; diese Bezeichnungen tauchen vor allem in den Aufgabenstellungen auf.

Die Infrastrukturen sind mit 1 beginnend aufsteigend durchnummeriert (analog zu den Gruppen). Damit es bei der Adressierung zu keinen Überschneidungen kommt, verwenden Sie nur IP-Adressen, die mit 10.⟨Infrastrukturnummer⟩ beginnen z.B. 10.1.0.1 ist eine IP-Adresse, die nur in Infrastruktur 1 verwendet werden darf. Das heißt, jeder Infrastruktur ist ein Subnetz 10.⟨Infrastrukturnummer⟩.0.0/16 zugeordnet.

Jede VM ist an ein *Management-Netz* angeschlossen. Zugang zu Ihren VMs erhalten Sie, über den Rechner `rnp@gruppeXY.rnp.lab.nm.ifi.lmu.de` Melden Sie sich zunächst per SSH an (`man ssh`) und ändern Sie Ihr Startpasswort. Von dort aus können Sie ihre VMs per SSH erreichen.

Beispiel: SSH-Aufruf für Gruppe 1 um auf `router4` zu gelangen:

```
$ ssh rnp@gruppe01.rnp.lab.nm.ifi.lmu.de
$ ssh root@router4
```

Auf den VMs wurde Public-Key Authentifizierung eingerichtet, deshalb benötigen Sie an dieser Stelle kein Passwort.

WARNUNG: Speichern Sie keine Daten auf den virtuellen Maschinen! Die virtuellen Maschinen werden u.U. automatisch neu aufgesetzt. Alle Daten, die zu diesem Zeitpunkt auf den VMs liegen gehen dabei verloren. Speichern Sie Daten deshalb in Ihrem CIP-Verzeichnis.

Indem Sie auf einem entfernten Rechner `tmux` benutzen können Sie mit nur einer SSH-Verbindung mehrere Konsolen nutzen. Eine tmux-Sitzung kann auch unterbrochen und später fortgesetzt werden. Bei `tmux` handelt es sich um einen Window-Manager für

[1] `https://openwrt.org/docs/start`

die Konsole. Zu dessen Fähigkeiten gehören auch copy-paste und mehrere Konsolen gleichzeitig anzeigen.

1.3.3 IProute2 – IP-Konfiguration

IProute2 ist eine Sammlung von Programmen für die Steuerung von Netzeinstellungen eines Rechners. Das prominenteste Tool aus dieser Sammlung ist `ip`. Die Funktionen von *ip* sind zahlreich und umfassen alles notwendige für die Konfiguration von (logischen) Verbindungen, IPv4-/IPv6-Adressen und Routen. Die man-page zu `ip` bietet einen Überblick über dessen Mächtigkeit.

Zu den grundlegenden, für das Praktikum relevante, Funktionen von `ip` gehören:

Aufruf	Erläuterung
ip link show	Alle Schnittstellen ausgeben
ip link set ethX up	Schnittstelle ethX aktivieren
ip address show	Alle Schnittstellen mit Adressen anzeigen
ip address add 192.168.1.1/24 \ ↪dev ethX	Schnittstelle ethX IP-Adresse 192.168.1.1 und Netzmaske 255.255.255.0 zuweisen (CIDR-Notation)
ip route show	Routingtabelle ausgeben
ip neighbor show	ARP-Tabelle ausgeben

Die Eingabe

```
# ip route show
```

gibt die aktuelle Routing-Tabelle auf der Kommandozeile aus.

Um eine Route in die Tabelle einzufügen benutzt man den Befehl

```
# ip route add <ZIEL> via <GATEWAY>

z.B.:
# ip route add 192.168.1.0/24 via 192.168.1.1
```

Das Beispiel zeigt einen Befehl, so dass alle Pakete, die ein Ziel im Subnetz 192.168.1.0/24 haben, zum Rechner 192.168.1.1 gesendet werden. Lässt man die Längenangabe für die Netz-ID (`/24`) weg, so wird eine Host-Route (mit signifikanten 32-Bit) hinzugefügt.

Um eine Route zu löschen, verwendet man

```
# ip route del <ZIEL>

z.B.:
# ip route del 192.168.1.0/24
```

ACHTUNG:
Ein Linux Kernel leitet normalerweise keine IP-Pakete weiter. Die Weiterleitung von IPv4 kann über das proc-Dateisystem aktiviert werden, indem der Wert in

`/proc/sys/net/ipv4/ip_forward` auf 1 gesetzt wird (Standardwert ist 0). Benutzen Sie dazu den Befehl:

```
# echo 1 > /proc/sys/net/ipv4/ip_forward
```

Unter [WWW 08] finden Sie eine vollständigere Dokumentation mit weiteren Erläuterungen und Beispielen.

1.3.4 tcpdump – Datenanalyse

Das Programm `tcpdump` zeigt Informationen aus den Protokollheadern eines jeden gesendeten oder empfangenen Pakets. Startet man `tcpdump` ohne Argumente, so wird der gesamte Datenverkehr aller Schnittstellen eines Rechners analysiert und optisch aufbereitet. Damit Sie auch sicher nur Ihren eigenen Datenverkehr analysieren achten Sie beim Aufruf von `tcpdump` darauf, dass Sie stets einen entsprechenden Filter für Ihr Subnetz benutzen. Zur optischen Aufbereitung von `tcpdump` gehört auch das Auflösen von IP-Adressen und Port-Nummern zu Rechner- und Port-Namen. In diesem Praktikum ist diese Anwendung von `tcpdump` nicht der Normalfall. Statt dessen werden meist folgende Optionen verwendet:

Option	Argument	Erläuterung
-n	*keine*	verhindert das Auflösen von numerischen Werten zu Namen
-i	Schnittstelle	beschränkt die Analyse auf eine Schnittstelle
-e	*keine*	aktiviert die Ausgabe des Schicht 2 Headers

Zusätzlich zu Optionen implementiert `tcpdump` eine Filtersprache, die es ermöglicht nur bestimmte Rahmen/Pakete/Segmente zu analysieren. Alle Optionen, sowie die Filtersprache werden in der manpage und unter [WWW 09] erläutert. `tcpdump` wird mit der Tastenkombination Strg + c beendet.

1.4 Aufgaben

A100 Adressierung und Wegewahl (Theorie)

i) Beschreiben Sie kurz die Bedeutung des Felds TTL im IPv4-Header und erklären Sie dabei, wie es von Schicht 3-Komponenten benutzt wird!

ii) Erklären Sie kurz das Verfahren der Unterteilung des IPv4-Adressraums in Klassen!

iii) Welche Probleme / Nachteile wurden durch die Einführung von CIDR behoben?

iv) Erstellen Sie analog zu Abbildung 1.6 einen Automaten zur Wegewahl für CIDR!

A101 Topology Discovery

Kapitel 1.3.2 beschreibt die virtuelle Infrastruktur, die Ihnen zur Verfügung steht und wie Sie Zugang zu Ihren virtuellen Maschinen erlangen.

Erstellen Sie einen Netzplan (siehe Kapitel 1.1.2) der Ihnen zur Verfügung stehenden virtuellen Maschinen!

i) Konfigurieren Sie Ihre virtuellen Maschinen mit IPv4 Adressen aus Ihrem Subnetz! Benutzen Sie dafür den Befehl `ip`! (Anmerkung: `ip help`)

ii) Ermitteln Sie Verbindungen zwischen zwei VMs, indem Sie mittels `ping`-Befehl Daten zwischen diesen hin und her schicken! (Anmerkung: `man ping`)

iii) Erstellen Sie einen Netzplan, der Ihre Konfigurationen und Ergebnisse wiederspiegelt!

A102 Fehlerdiagnose mit tcpdump

Benutzen Sie in dieser Aufgabe `tcpdump` um ICMP "echo request" PDUs sichtbar zu machen. Starten Sie dazu einen ping zwischen `pc1` und `router1`. Starten Sie nun auf einem der beiden Rechner `tcpdump`, um die ICMP PDUs mitlesen zu können.

Erläutern Sie die Ausgabe von `tcpdump` wenn ...

i) ... Sie **nicht** die Option -e angeben.

ii) ... Sie die Option -e angeben.

iii) Führen Sie nun auf `pc1` den Befehl `ifup eth1` aus. Damit erhält der Rechner die IP-Adresse *172.16.1.100*. Vergeben Sie auf dem entsprechenden Interface von `router1` die IP-Adresse *172.16.1.1*. Wiederholen Sie den ping-Vorgang mit diesen Adressen.

iv) Verwenden Sie tcpdump um den Unterschied zwischen den IP-Adressen zu sehen und erklären Sie warum keine Antwort ankommt.

v) Korrigieren Sie den Fehler und zeigen Sie dass der ping-Vorgang nun erfolgreich ist.

A103 Statisches Routing

Konfigurieren Sie nun die VMs, um Nachrichten über mehrere Hops zu transportieren. *Hinweis:* Alle Teilaufgaben sollen so gelöst werden, dass IP-Pakete „hin und zurück", d.h. in beide Richtungen zwischen Sender und Empfänger vermittelt werden.

Um die Routing-Tabelle eines Rechners einsehen und administrieren zu können, benutzen Sie das Werkzeug `ip`:

i) Konfigurieren Sie **Host**-Routen, so dass pc1 und pc2 Daten austauschen können! Weisen Sie mittels ICMP echo request/reply nach, dass dies der Fall ist! Nehmen Sie die nach erfolgreicher Konfiguration geltenden Routing-Tabellen der beteiligten VMs in Ihre Ausarbeitung auf! Einzelne exemplarische Tabellen reichen, sofern daraus der Ablauf ersichtlich wird.

ii) Ersetzen Sie die Host-Routen auf pc1 und pc2 durch default Routen!

iii) Ersetzen Sie die Host-Routen auf den Routern durch Routen in die jeweiligen Subnetze von pc1 und pc2!

iv) Starten Sie auf pc1 einen traceroute nach pc2. Erläutern Sie anhand eines tcpdump-Mitschnitts die Funktionsweise von traceroute!

v) Konfigurieren Sie Ihr Routing so, dass Daten von pc1 an pc2 *immer* über router4 vermittelt werden und Daten von pc2 an pc1 *nie* über router4 vermittelt werden! Zeigen Sie, dass Ihre Konfiguration funktioniert, indem Sie entsprechende traceroute-Ausgaben erzeugen. Zeichnen Sie einen **gerichteten** Netzplan für Ihren Aufbau.

◇

Literatur

[Garr 08] GARRELS, MACHTELT: *Bash Guide for Beginners*. The Linux Documentation Project Guide, Dezember 2008, http://tldp.org/LDP/Bash-Beginners-Guide/html/Bash-Beginners-Guide.html .

[RFC 1349] ALMQUIST, P.: *Type of Service in the Internet Protocol Suite*. RFC 1349 (Proposed Standard), Juli 1992, http://www.ietf.org/rfc/rfc1349.txt . Obsoleted by RFC 2474.

[RFC 1519] FULLER, V., T. LI, J. YU und K. VARADHAN: *Classless Inter-Domain Routing (CIDR): an Address Assignment and Aggregation Strategy*. RFC 1519 (Proposed Standard), September 1993, http://www.ietf.org/rfc/rfc1519.txt . Obsoleted by RFC 4632.

[RFC 2474] NICHOLS, K., S. BLAKE, F. BAKER und D. BLACK: *Definition of the Differentiated Services Field (DS Field) in the IPv4 and IPv6 Headers*. RFC 2474 (Proposed Standard), Dezember 1998, http://www.ietf.org/rfc/rfc2474.txt . Updated by RFCs 3168, 3260.

[RFC 3168] RAMAKRISHNAN, K., S. FLOYD und D. BLACK: *The Addition of Explicit Congestion Notification (ECN) to IP*. RFC 3168 (Proposed Standard), September 2001, http://www.ietf.org/rfc/rfc3168.txt .

[RFC 3260] GROSSMAN, D.: *New Terminology and Clarifications for Diffserv*. RFC 3260 (Informational), April 2002, http://www.ietf.org/rfc/rfc3260.txt .

[RFC 4632] FULLER, V. und T. LI: *Classless Inter-domain Routing (CIDR): The Internet Address Assignment and Aggregation Plan*. RFC 4632 (Best Current Practice), August 2006, http://www.ietf.org/rfc/rfc4632.txt .

[RFC 791] POSTEL, J.: *Internet Protocol*. RFC 791 (Standard), September 1981, http://www.ietf.org/rfc/rfc791.txt . Updated by RFC 1349.

[WWW 08] *IProute2 Homepage*. The Linux Foundation, 2008, http://www.linuxfoundation.org/en/Net:Iproute2 .

[WWW 09] *TCPDUMP/LIBPCAP Projekt Homepage*, Januar 2009, http://www.tcpdump.org .

INSTITUT FÜR
INFORMATIK
DER LUDWIG-MAXIMILIANS-UNIVERSITÄT MÜNCHEN

LS Prof. Kranzlmüller

Praktikum Rechnernetze

Kapitel 2: Signalisierung und Rahmenbildung

MNM
TEAM
Munich Network Management Team

2 Signalisierung und Rahmenbildung

Inhaltsverzeichnis

2.1 Bitübertragungsschicht (OSI-Schicht 1) 21
2.1.1 Repeater und Hubs 22
2.1.2 Wavelength Division Multiplexer 22
2.2 Sicherungsschicht (OSI-Schicht 2) 23
2.2.1 Vergleich von Komponenten der Schichten 1 und 2 23
2.2.2 Topologien 24
2.3 Ethernet 25
2.4 Virtuelle Topologien 26
2.4.1 Monitoring-Port Konzept 26
2.5 TUN/TAP Devices 27
2.5.1 Virtuelle Schnittstellen unter Linux 28
2.6 Scapy 29
2.6.1 Installation auf der Infrastruktur 29
2.6.2 Usage 29
2.7 Aufgaben 31

Dieser Abschnitt beschäftigt sich mit der Bitübertragungsschicht und der Sicherungsschicht, den Schichten 1 und 2 im ISO-OSI Referenzmodell. In anderen Modellen sind diese Schichten zu einer “Netzzugangsschicht” zusammengefasst.

2.1 Bitübertragungsschicht (OSI-Schicht 1)

Bei jeder Interaktion zwischen zwei Rechnern müssen die Daten früher oder später eine physische Distanz überbrücken. Die binären Daten eines Rechners werden für den physischen Transport aufbereitet und anschließend über ein Medium an einen anderen Rechner übertragen. Kann der andere Rechner aus dem Empfangenen die ursprünglichen Daten rekonstruieren, ist es gelungen binäre Daten von einem Rechner an einen anderen zu übertragen.

Protokolle der Bitübertragungsschicht beschäftigen sich mit genau diesem Problem. Sie legen fest welches **Medium** benutzt wird und wie binäre Daten (Bits) als **Signale** auf das physische Medium **moduliert** werden. Dazu gehören mehrere Aspekte, die sicherstellen, dass alle Endpunkte auf die selbe Art und Weise Daten übermitteln und interpretieren. Diese Aspekte lassen sich in vier Gruppen unterteilen:

Physikalische Aspekte umfassen Eigenschaften des Mediums und der verwendeten Signale.

Mechanische Eigenschaften spezifizieren u.A. die Bauform der Anschlüsse an das Medium.

Funktionale Spezifikationen definieren die Benutzung des Mediums, z.B. Pin-Belegung und Takt.

Prozedurale Beschreibungen enthalten Elementarereignisse und deren Bedeutung, z.B. den genauen Ablauf zur Übertragung einer SDU.

Heute handelt es sich bei dem Medium meist um Kupfer, Lichtwellenleiter oder "Luft". Die Signale werden in der Regel so gewählt, dass sie deutlich voneinander unterscheidbar sind, da Signale durch Störeinflüsse leicht verfälscht werden können. Eine SDU auf Schicht 1 muss nicht genau ein Bit sein. Ein Schicht 1 Protokoll kann auch die parallele Übertragung von mehreren Bits gleichzeitig spezifizieren.

Durch äußere Störeinflüsse können Amplitude, Frequenz und Phase eines Signals bei der Übertragung verändert werden. Komponenten der Schicht 1 verarbeiten Signale dahingehend, dass eingehende Signale als Signalzustände entsprechend der Spezifikation aufgefasst werden (Diskretisierung). Die Umwandlung in Bits ist ein weiterer Arbeitsschritt, der in der Regel nur auf Komponenten, die auch eine Schicht 2 implementieren, durchgeführt werden muss. Gängige Schicht 1 Komponenten sind **Repeater**, Multiport-Repeater (**Hubs**) und Wavelength Division Multiplexer (**WDMs**).

2.1.1 Repeater und Hubs

Ein Repeater verfügt über genau zwei Anschlüsse. Ein eingehendes Signal auf einem Anschluss wird verstärkt auf dem anderen Anschluss ausgegeben. Repeater können eingesetzt werden, um das Problem der Dämpfung zu überwinden und Signale über längere Distanzen zu übertragen, als es die Sendeleistung des ursprünglichen Senders erlaubt. Die logische Weiterentwicklung des Repeaters ist der Hub. Dieser verfügt über mehrere Anschlüsse und gibt ein eingehendes Signal an allen anderen Anschlüssen verstärkt wieder aus.

2.1.2 Wavelength Division Multiplexer

Implementieren optische Sendestationen das selbe Schicht 1 Protokoll, so verwenden sie meist die selben Wellenlängen zur Signalisierung. Treffen diese Signale in einem gemeinsamen Medium aufeinander, entstehen Überlagerungen (Kollisionen), so dass kein Empfänger die ursprünglichen Daten rekonstruieren kann. Bei einem Aufbau, in dem mehrere Sender Signale auf dem selben Medium versenden, so dass Kollisionen entstehen können, spricht man von einer Kollisionsdomäne.

WDMs bilden mehrere eingehende optische Signale auf unterschiedliche, disjunkte Bereiche des Farbspektrums (Kanäle) einer ausgehenden Leitung ab (Multiplex). Dadurch

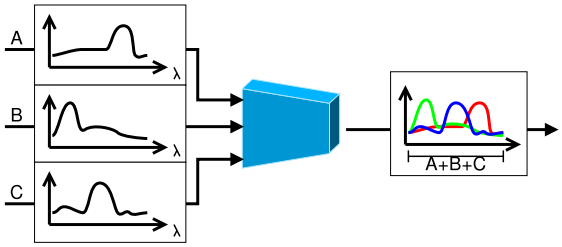


Abb. 2.1: Funktionsweise eines WDMs, der drei eingehende Signale auf ein gemeinsames Medium moduliert

können alle eingehenden Signale gleichzeitig über eine einzelne ausgehende Leitung übertragen werden, ohne Kollisionen zu erzeugen. Abbildung 2.1 zeigt einen WDM, der drei eingehende Signale A, B und C auf eine ausgehende Leitung moduliert. Diese Technik setzt man häufig ein, wenn der Aufwand zusätzliche Leitungen zu verlegen hoch ist. In Abbildung 2.1 sind A, B und C optische Signale der selben Schicht 1 Implementierung (z.B. 1 Gbps Ethernet, Monomode) und verwenden deshalb die selbe Wellenlänge für die Datenübertragung. Im WDM werden die Wellenlängen der eingehenden Signale modifiziert, so dass keine Überlagerung mehr stattfindet wenn die Signale in der ausgehenden Leitung aufeinander treffen.

Am anderen Ende der Leitung empfängt ein Demultiplexer das zusammengesetzte Signal. Dieser trennt das empfangene Signal auf, moduliert die Teilsignale zurück auf ihre ursprüngliche Form und sendet die Signale auf separaten Leitungen weiter.

2.2 Sicherungsschicht (OSI-Schicht 2)

Zu den Hauptaufgaben der Sicherungsschicht gehört Rahmenbildung (bzw. Blockbildung). Darunter versteht man die Gruppierung von Bits zu logischen Einheiten, den PDUs der Schicht 2 (Rahmen oder Blöcke, bzw. engl. frames oder blocks). Eine Basiskomponente der Schicht 2 ist die Bridge (bzw. “Brücke”). Eine Bridge ist eine Schicht 2 Komponente, die zwei Teilnetze (mit möglicherweise verschiedenen Übertragungstechniken) miteinander verbindet, also eine Brücke dazwischen bildet. Dazu muss sie eingehende Signale **interpretieren** und zu Rahmen zusammensetzen. Erst der vollständige Rahmen wird in das andere Teilnetz übertragen. Diese Technik heißt “store and forward”, da die Bridge eingehende Daten (Bits) speichert (store), bis der Rahmen vollständig empfangen wurde und erst im Anschluss daran den Rahmen weiterleitet (forward). Eine Bridge mit mehr als zwei Anschlüssen heißt Switch oder Multiport-Bridge.

2.2.1 Vergleich von Komponenten der Schichten 1 und 2

Aufgrund der Hauptaufgaben der Schichten 1 und 2 ergeben sich unterscheidende Merkmale der Komponenten dieser Schichten: Repeater, Hubs, WDM etc. auf der Schicht 1,

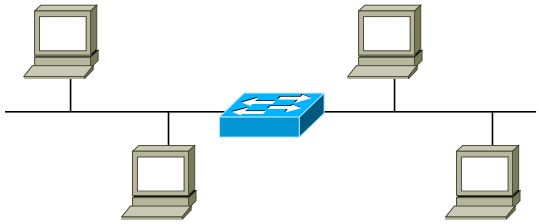


Abb. 2.2: Zwei Netze mit Bustopologie, verbunden über eine Bridge

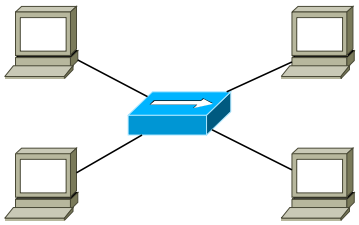


Abb. 2.3: Netz mit Sterntopologie

Bridges und Switches auf der Schicht 2.

Im Gegensatz zu einem WDM verarbeitet eine Bridge nicht einzelne eingehende Signale, sondern eingehende Rahmen. Die Bridge muss demnach in der Lage sein aus gespeicherten Informationen Rahmen zu bilden. Der Einsatz von store and forward ermöglicht es der Bridge unterschiedliche Bitübertragungstechniken für die beiden LANs zu verwenden, wogegen die Anschlüsse eines WDM den Festlegungen der physikalischen Signale entsprechen. Der Aufbau in Abbildung 2.2 erlaubt es unterschiedliche Schicht 1 Implementierungen “links” und “rechts” der Bridge zu nutzen.

2.2.2 Topologien

Neben der Rahmenbildung spezifizieren Schicht 2 Protokolle auch die Übertragung von Rahmen. Dazu gehören Vielfachzugriffsverfahren, die den Zugriff auf Schicht 1 bzw. auf gemeinsam genutzte Medien steuern, sowie die Übertragung von Rahmen zu Schicht 2 Endpunkten. Durch die vielfältigen Komponenten und Funktionen der Schichten 1 und 2 ergeben sich verschiedene Möglichkeiten einen physischen Aufbau eines Netzes (LAN) zu realisieren.

Abbildung 2.2 zeigt auf jeder Seite der Bridge ein Netz mit **Bustopologie**. Dabei sind mehrere Schicht 2 Endpunkte mit einem gemeinsam genutzten Medium verbunden. Ein Problem der Bustopologie ist die aufwändige Wartung. Tritt ein Hardwaredefekt auf, so kommt meistens das gesamte LAN zum Erliegen. Das Aufspüren der defekten Hardware ist schwierig, da jede Komponente für das Problem verantwortlich sein könnte. Das gemeinsam genutzte Kabel erstreckt sich meist über mehrere Räume oder auch Stockwerke und ist sehr aufwändig auszutauschen.

Eine andere Topologie, die bei diesem Problem hilft, ist die **Sterntopologie**. An die Stelle des gemeinsam genutzen Mediums tritt ein zentraler Hub (vgl. Abbildung 2.3). Da der Hub eingehende Signale auf alle angeschlossene Kabel repliziert, verhält sich ein Netz mit Sterntopologie genauso wie ein Netz mit Bustopologie; der Charakter des gemeinsam genutzten Mediums bleibt für die Signalübertragung erhalten. Deshalb kann an jeden Anschluss eines Hubs ein ganzes Teilnetz mit Bustopologie angeschlossen werden.

Der Vorteil des Hubs liegt darin, dass ein defektes Kabel an einem Anschluss nicht automatisch zu einem Zusammenbruch des gesamten Netzes führt. Außerdem kann an zentraler Stelle das fehlerhafte Kabel ermittelt werden. Üblicherweise ist an jeder Leitung eines Hubs ein einzelner Rechner angeschlossen, wodurch die Fehlerlokalisierung weiter vereinfacht wird. Der Defekt eines Kabels trennt so lediglich eine einzige Leitung (bzw. einen einzelnen Rechner) vom LAN, anstatt das gesamte LAN zum Erliegen zu bringen. Fällt der Hub aus, kann dieser leichter ausgetauscht werden, als ein Kabel, das durch mehrere Räume und Stockwerke verlegt wurde. Ersetzt man den Hub durch einen Switch, kann der Datenverkehr mit Bezug auf die Eigenschaften von Rahmen (Header) gesteuert werden. Ein Switch kann durch die Interpretation der eingehenden Informationen entscheiden eingehende Rahmen über wenige bestimmte Leitungen weitergeben, anstatt den Rahmen auf jedem Port zu replizieren.

2.3 Ethernet

Infolge der wachsenden Bedeutung der lokalen Vernetzung von Arbeitsplatzrechnern wurden zwischen 1972 und 1976 am *Xerox Palo Alto Research Center* die technologischen Grundlagen für ein gleichermaßen leistungsfähiges und “idiotensicheres” Local Area Network geschaffen. Dieses neue Local Area Network nannte man **Ethernet**, in Anspielung auf jenen geheimnisvollen “Lichtwellenäther”, welchen die Physiker des 19. Jahrhunderts so verzweifelt gesucht haben. Heute wird Ethernet formal als IEEE-Standard 802.3, CSMA/CD: Protokoll und physische Übertragungstechniken [IEEE 802.3], verwaltet und weiterentwickelt.

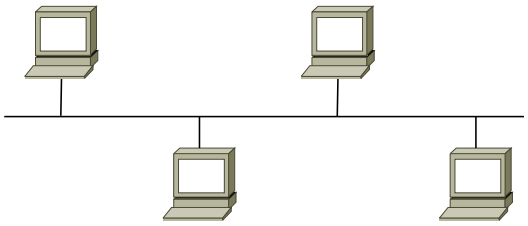


Abb. 2.4: Ethernet mit Bustopologie

Ethernet basiert in seiner ursprünglichen Form auf einer Broadcast-Technik, bei der alle Komponenten an das selbe Medium angeschlossen sind, wie in Abbildung 2.4 dargestellt. Sendet eine Komponente Daten, so werden diese von jeder anderen Komponente

empfangen. Senden mehrere Komponenten gleichzeitig, entsteht eine Datenkollision, so dass letztendlich keine Daten übertragen werden können. Bei einer Kollision treffen die Signale im Medium aufeinander, wodurch Interferenzen entstehen, so dass die ursprünglichen Signale nicht mehr unterscheidbar sind (siehe Kapitel 2.1). Aus diesem Grund benutzt Ethernet CSMA/CD, ein Verfahren um die Nutzung des gemeinsamen Mediums zu koordinieren. Die meisten heutigen Ethernet-Netze sind Sterntopologien, bei denen jeder Endpunkt (z.B. Rechner) exklusiv mit einem Switch-Port verbunden ist.

2.4 Virtuelle Topologien

Endeinrichtungen (Rechner) werden in der Praxis bestimmten Aufgaben oder Rollen einer Organisation zugeordnet, statt den Gegebenheiten der Vernetzung: Es ist z.B. wünschenswert, die Rechner nach Abteilungen bzw. Bereichen zu gruppieren. Oft entspricht dabei die physische Topologie, die durch die verlegten Medien (“Kabel”) gegeben ist, nicht den Nutzungsanforderungen eines LANs: z.B. werden Server in gemeinsam genutzten Server-Räumen untergebracht, in denen sich alle Server die selbe Leitung aus dem Raum hinaus teilen. Meist ist es jedoch so, dass sensible Daten nicht in andere LANs als dem abteilungseigenen LAN gelangen dürfen.

Hierzu können sogenannte *virtuelle LANs* (VLAN) benutzt werden, die eine logische LAN-Topologie auf eine physische aufbringen. VLANs können auf mehrere Arten erzeugt werden: durch die Gruppierung von Ports an einem Switch, durch Gruppierung von MAC-Adressen der zu einer Gruppe gehörenden Rechner und durch eine entsprechende Markierungen (*engl. tagging*) der gesendeten Rahmen. Im Praktikum liegt dabei der Schwerpunkt auf der zuletzt genannten Technik.

Ein wichtiger Standard in diesem Kontext ist der IEEE-Standard 802.1q [IEEE 802.1q], “Virtual LANs”. Dieser definiert das Anlegen, Betreiben und Verwalten von (mehreren) virtuellen LAN-Topologien innerhalb eines physischen LANs. Dazu wird jeder Rahmen einer virtuellen Infrastruktur mit einer für diese Infrastruktur eindeutigen Nummer (VLAN Identifier, VLAN-ID) in einem Feld (VLAN-Tag) im Ethernet-Header markiert. Netzkomponenten, die auf Schicht 2 operieren, können anhand der VLAN-ID Rahmen virtueller Topologien unterscheiden und unterschiedlich behandeln.

2.4.1 Monitoring-Port Konzept

Administrierbare Switches bieten meist die Funktion eines Monitoring-Ports. Dabei wird ein bestimmter Port des Switches ausgewählt, auf dem der Switch jeden Rahmen repliziert, der auf einem der anderen Switch-Ports empfangen wird. Diese Funktion ermöglicht es, jeden Rahmen, der vom Switch verarbeitet wird, an zentraler Stelle zu sammeln. Da die Übertragungsrate eines Monitoring-Ports meist deutlich kleiner ist als die aller anderen Ports zusammen, kann ein Monitoring-Port ab einer gewissen Auslastung nicht alle Rahmen replizieren. Aus diesem Grund werden Monitoring-Ports häufig nur zur Fehleranalyse eingesetzt. Das Konzept des Monitoring-Ports kann auch virtualisiert umgesetzt werden.

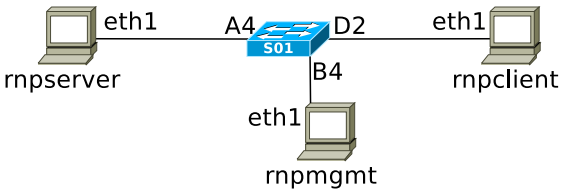


Abb. 2.5: Szenario für Management-Port Beispiel

Abbildung 2.5 zeigt ein Beispiel in dem die Rechner rnpserver und rnpclient über (rnpserver-eth1, S01-A-4) und (S01-D-2, rnpclient-eth1) verbunden sind. Außerdem ist der Rechner rnpmgmt mit dem Switch S01 verbunden. Wird der Switch-Port S01-B-4 als Management-Port konfiguriert, so empfängt der Rechner rnpmgmt an seiner Schnittstelle eth1 alle Rahmen, die rnpserver und rnpclient austauschen. Dies ermöglicht es die Interaktion dieser beiden Rechner zu überwachen, ohne direkten Zugang zu haben.

Das Einrichten eines Management-Ports ist eine spezielle Funktion, die von der Konfigurationssoftware der Switches bereitgestellt wird.

2.5 TUN/TAP Devices

Linux bietet seit Kernel-Version 2.2 die Möglichkeit virtueller Treiber-Schnittstellen an, sogenannte TUN/TAP Devices. TUN/TAP Devices existieren nur im Kernel und haben im Vergleich zu gewöhnlichen Schnittstellen keine physische Komponente. Falls das Betriebssystem Daten an ein TUN/TAP Device sendet, wird die Nachricht nicht an das physische Device, sondern an eine Benutzeranwendung, die über ein File-Descriptor mit dem TUN/TAP Device verbunden ist, weitergegeben. Was dann tatsächlich mit den Daten passiert, bleibt der Anwendung überlassen. Ein typisches Einsatzgebiet ist Tunneling, wie es beispielsweise in Virtual Private Networks (VPNs) der Fall ist. Durch den Einsatz von TUN Devices empfängt die VPN-Anwendung alle IP-Pakete, verschlüsselt deren Payload und umrahmt das ursprüngliche IP-Paket in ein weiteres IP-Paket mit aktualisierten Header-Informationen. Der Unterschied zwischen TUN und TAP Interfaces liegt darin, dass TAP Devices auf Schicht 2 und TUN Devices auf Schicht 3 operieren. Diese Unterscheidung ist wichtig, denn je nach Anwendungsfall ergeben sich entsprechende Anforderungen. Für VPN-Anwendungen bspw. sind TUN Devices auf Schicht 3 ausreichend. Eine ausführliche Dokumentation findet sich in der Linux-Kernel Dokumentation[1].

Im Rahmen dieses Praktikums setzen wir TUN/TAP Devices dazu ein, um unseren eigenen Netzwerk-Stack zu implementieren. Ein einfaches Beispiel findet sich in der beigelegten Mini-Anwendung[2]). Bevor ein TUN / TAP Device genutzt werden kann,

[1] `https://git.kernel.org/pub/scm/linux/kernel/git/torvalds/linux.git/tree/Documentation/networking/tuntap.txt`

[2] siehe `tuntap.c`

erzeugen wir zunächst ein virtuelles Device. Hierzu nutzen wir das allseits bekannte Tool iproute (`ip`).

```
$ ip tuntap help
   Usage: ip tuntap { add | del } [ dev PHYS_DEV ]
             [ mode { tun | tap } ] [ user USER ] [ group GROUP ]
             [ one_queue ] [ pi ] [ vnet_hdr ]

   Where: USER  := { STRING | NUMBER }
             GROUP := { STRING | NUMBER }
```

Nachdem ein physisches Device erzeugt wurde, setzen wir das Device zunächst auf den Status `up` und geben ihm anschließend ein Netz bzw. eine fixe Host-Adresse. Mit der richtigen Konfiguration wird jeglicher Datenverkehr durch das das TUN/TAP Device geleitet.

2.5.1 Virtuelle Schnittstellen unter Linux

Allgemein kann man sagen: Virtualisierung ist die Abstraktion von starren, beschränkenden Randbedingungen eines Systems zu konfigurierbaren Eigenschaften. Im Fall von virtuellen Schnittstellen bedeutet Virtualisierung, dass man die Anzahl der Schnittstellen eines Rechners verändern kann. Freilich lassen sich dadurch nicht mehr Kabel mit einem Rechner verbinden als physische Schnittstellen vorhanden sind. Trotzdem erhöhen sich die Möglichkeiten im Management.

Im Rahmen dieses Praktikums bieten virtuelle Schnittstellen den Mehrwert, dass man über eine physische Verbindung mehrere logische Verbindungen betreiben kann (vgl. Abbildung 2.6).

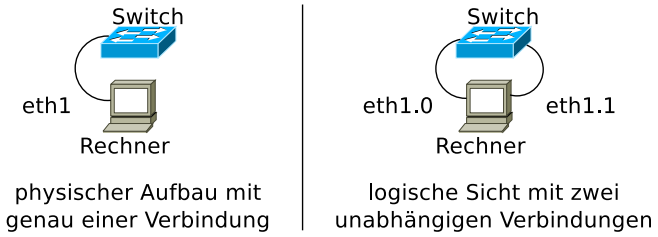


Abb. 2.6: Zwei logische Schnittstellen teilen sich eine Leitung

Jeder virtuellen Schnittstelle kann außer eigenen IP-Adressen auch eine eigene VLAN-ID zugewiesen werden. Es ist somit möglich, mit nur einer physischen Verbindung einen Rechner in mehrere virtuelle LANs einzubinden.

Es ist möglich einer einzelnen Schnittstelle mehrere IPs zuzuweisen und so einen Rechner in mehrere Subnets (z.B. 192.168.1/24 und 192.168.2/24) einzubinden. Der wesentliche Unterschied zum Einsatz virtueller Schnittstellen und VLANs liegt darin, dass virtuelle Schnittstellen bereits auf OSI-Schicht 2 implementiert werden, eine Trennung auf IP-Ebene jedoch eine Funktion von OSI-Schicht 3 ist.

Anlegen virtueller Schnittstellen mit VLAN-ID:
Das Anlegen von virtuellen Schnittstellen ist ebenfalls eine Funktion, die mit dem Befehl `ip` realisiert werden kann. Die Syntax zum Anlegen einer VLAN-Schnittstelle ist:

```
# ip link add link <pNIC> name <vNIC> type vlan id <VLAN-ID>
mit
   pNIC := Name der physischen Schnittstelle
   vNIC := Name der virtuellen Schnittstelle

z.B.:
# ip link add link eth0 name eth0.100 type vlan id 100
```

Zum Entfernen einer virtuellen Schnittstelle benutzen Sie:

```
# ip link del dev <vNIC>

z.B.:
# ip link del dev eth0.100
```

2.6 Scapy

Scapy ist ein Python Framework zur Inspektion und Manipulation von Paketen. Es erlaubt Ihnen Pakete vieler bereites implementierter Protokolle zu protokollieren, zu dekodieren, aus `pcap`-Dateien zu lesen, zu erstellen sowie diese zu versenden und vieles mehr. Scapy wurde auch für schnelles Prototyping entwickelt und benutzt Defaultwerte, die funktionieren.

Viele Aufgaben anderer Tools können von Scapy übernommen werden, z.B. Scanning, Traceroutes, Unit-Tests, Netzwerkerkennung und verschiedene Angriffe. Außerdem können Sie auch ungültige Rahmen versenden, eigene 802.11 Rahmen einbauen und verschiedene Techniken kombinieren (VLAN hopping+ARP cache poisoning, VoIP-Dekodierung auf einem WEP-verschlüsselten Kanal, etc.).

2.6.1 Installation auf der Infrastruktur

```
// On Debian
apt-get install python3-scapy

// On OpenWRT
opkg update; opkg install scapy
```

2.6.2 Usage

Eine kleine Übersicht:

- Verschiedene Protokolle und Header sind als Klassen implementiert, wie `IPv6` oder `ICMP`.
- Protokolle können ineinander geschachtelt werden mit dem Slash-Operator, z.B.: `IPv6()/TCP()` erstellt ein TCP-over-IPv6 Paket

- Pakete können mit den Methoden `show` und `show2` angezeigt werden, für das Senden bzw. Senden und Empfangen siehe `send()`, `sendp()`, `sr()`, `sr1()` und `srp()`.
- Die meisten Headeroptionen können verändert werden. Setzen Sie diese entweder im Konstruktor oder über Membervariablen, z.B.: `p = IP(ttl=64)`
- Nicht alle Optionen müssen angegeben werden, Scapy befüllt solche Werte mit Defaults
- Eine Liste aller Funktionen bekommen Sie über die Funktion `ls()`
- Für weitere Informationen und eine Demonstration, siehe `https://scapy.net/`

2.7 Aufgaben

Hinweis: vergessen Sie nicht Ihre Konfiguration in Netzplänen zu dokumentieren und auch die relevanten Ausgaben der verwendeten Programme zu übernehmen!

A200 Address Resolution and Neighbor Discovery (Theorie)

Das RFC 826 definiert das Address Resolution Protocol (ARP).
In RFC 4861 wird das Neighbor Discovery Protocol (NDP) spezifiziert.

i) Wozu wird ARP eingesetzt? Was ist der Unterschied zu NDP?

ii) Beschreiben Sie den Aufbau einer ARP-PDU und erläutern Sie die Bedeutung der einzelnen Felder!

iii) Welche unterschiedlichen ARP-PDUs gibt es? Welche NDP-PDUs gibt es?

iv) Wie lang (in Bytes) ist eine ARP-PDU in einem Netz in dem IPv4 und Ethernet eingesetzt werden?

v) Wie lang (in Bytes) ist eine Neighbor Solicitation Nachricht?

vi) Das RFC 826 spricht von einer Tabelle (table), deren Implementierung meist als ARP-Cache bezeichnet wird. Was soll laut RFC mit einer Ethernet-SDU passieren, wenn kein Eintrag zur Ziel-IP-Adresse in der Tabelle gefunden wird?

A201 VLANs nach IEEE 802.1q

Virtuelle Infrastrukturen dienen dazu Netze anzulegen und anpassen zu können, ohne physisch an den Geräten arbeiten zu müssen. Dies ist insbesondere in großen, weniger übersichtlichen Infrastrukturen von Vorteil.

Nachdem Sie in Aufgabe `A101` die Topologie der virtuellen Infrastruktur vollständig rekonstruiert haben, ändern Sie diese im Folgenden.

i) Modifizieren Sie die Topologie Ihrer virtuellen Infrastruktur so, dass die Router 1,2 und 3 ausschließlich Switches (Bridges) sind. Fügen Sie dieser Bridge alle Interfaces bis auf eth0 hinzu. (Hinweis: Benutzen Sie `ip link`, siehe Referenz[3])

   Die PCs 1–3 sowie Router 4 sind Hosts, die in *einem* (Sub-)Netz gemäß der Baum-Topologie in Abbildung 2.7 verbunden sind. Es genügt hierbei, wenn Sie überflüssige Links als `down` markieren.

ii) Testen Sie ihre Verbindungen zwischen den PCs 1–3 sowie Router 4 mittels `ping`. Router 1–3 haben per Definition von Bridges *keine* IP-Adresse. Vergeben Sie IPv4-Adressen aus Ihrem Adressraum.

[3] `https://wiki.archlinux.org/title/Network_bridge`

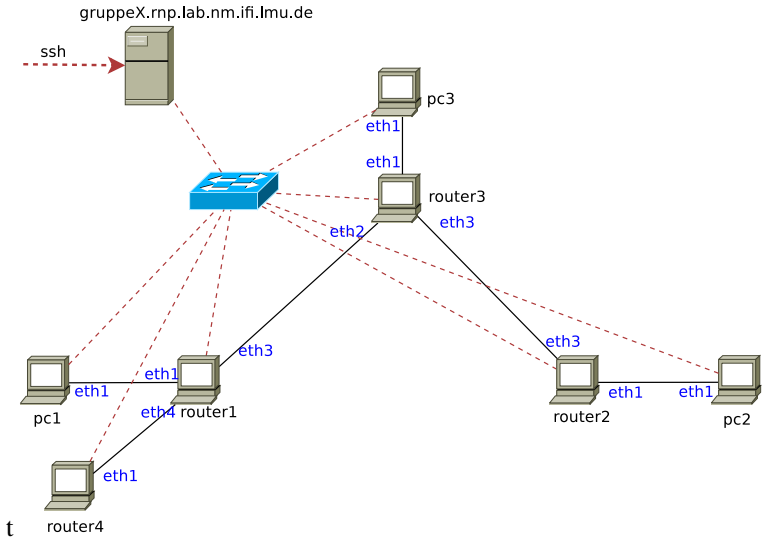


Abb. 2.7: Baum-Topologie

iii) Derzeit befinden sich alle Hosts im selben Netz. In dieser Aufgabe soll das Netz in zwei logisch getrennte VLANs (Schicht 2) aufgeteilt werden. Ziel ist es, dass Router 4 und pc3 in einem VLAN sowie pc2 und pc1 in einem anderen VLAN voneinander isoliert sind. Die *Schicht 3*-Adressen bleiben davon unberührt. Das Ergebnis soll sein, dass ausschließlich innerhalb des VLANs kommuniziert werden kann. Heißt also, dass bspw. Router 4 nur noch mit pc3 kommunizieren kann, allerdings nicht mehr mit den PCs 1–2 (obwohl diese noch im selben Subnetz auf Schicht 3 liegen).

   Testen Sie Ihre Konfiguration mittels `ping` und weisen Sie durch Mitlauschen via `tcpdump` (auf einem der Router 1–3), nach, dass bei den ICMP-Anfragen anhand der VLAN-IDs innerhalb des VLANs kommuniziert wird.

`A202` Analyse von ARP und NDP

Verwenden Sie die gleiche Topologie wie in der vorherigen Aufgabe. Entfernen Sie alle VLANS, sofern welche vorhanden sind.

i) Erläutern Sie die Ausgabe von `ip neigh` auf pc1!

ii) Leeren Sie den ARP-Cache von pc1 und pc2! (Hinweis: `ip neigh`)

iii) Stellen Sie sicher, dass die Interfaces der PCs Link-Local IPv6-Adressen haben. (Hinweis: `ip link [down/up]`)

iv) Benutzen Sie `ping` und `ping6` um IP-Verkehr zwischen pc1 und pc2 zu erzeugen. Wie verändert sich die Ausgabe von `ip neigh`?

v) Beobachten Sie mit `tcpdump` die Auflösung von IP- zu MAC-Adressen. An wel-

che MAC-Adressen werden die Anfragen zur Adressauflösung jeweils geschickt? Handelt es sich um Broadcasts?

vi) Fügen Sie Ihren PCs zusätzlich IPv6-Adressen aus folgenden Netz hinzu: `2001:db8:<Gruppennummer>::/64`

vii) Wiederholen Sie den IPv6-Versuch mit den neu hinzugefügten globalen Adressen. Erläutern Sie die Unterschiede!

viii) Vergleichen Sie die Ausgabe von `tcpdump` mit `scapy`. Lassen Sie sich die Details eines ARP/Neighbor Solicitation (NS) Pakets mit `scapy` ausgeben.

ix) Verwenden Sie `scapy` um sowohl ein ARP- als auch ein NS-Paket von pc1 zu versenden. (Hinweis: `Ether` und `sendp()`)

Erhalten pc2 und pc3 diese Pakete? Wie reagieren diese auf die Anfragen? Betrachten Sie die Antworten, die pc1 erhält!

x) Senden Sie jeweils ein ARP Reply und ein Neighbor Advertisement (NA) Paket mit einer neuen MAC-Adresse von pc1 zu pc2 mit `scapy`. Wie verhält sich der Neighbor cache auf pc2 nach deren Eingang?

A203 Implementierung von ARP in C

Die Aufgabe ist eine eigenständige Implementierung des ARP-Protokolls (vgl. RFC 826). Ziel ist es, dass Sie auf Basis von TUN/TAP Devices den Datenverkehr innerhalb eines Netzes abfangen und auf ARP-Anfragen mittels des Tools `arping` eine semantisch und syntaktisch korrekte ARP-Antwort zurück senden.

i) Machen Sie sich mit der Funktionsweise von TUN/TAP Devices vertraut und prüfen Sie mittels der Mini-Applikation `tuntap.c`, ob Ihre Konfiguration korrekt funktioniert.

ii) Überlegen Sie sich ein geeignetes Interface für ARP-Anfragen bzw. -Antworten und dokumentieren es in einem Header-File. Lesen Sie hierzu RFC 826[4].

iii) Implementieren Sie ihr spezifiziertes Interface so, dass eine `arping` Anfrage korrekt beantwortet wird. Sie können für Ihre Antwort eine beliebige IP-Adresse bzw. MAC-Adresse vergeben.

Beispiel für eine `arping` Anfrage:

```
$ arping -I mytap <ip>
```

◇

[4] `https://tools.ietf.org/html/rfc826`

Literatur

[IEEE 802.1q] *IEEE Std 802.1q-2005*, Mai 2006.

[IEEE 802] *IEEE Std 802-2001*, Februar 2007.

[IEEE 802.3] IEEE 802-3 WORK GROUP: *IEEE Std 802.3-2005*, Dezember 2005.

INSTITUT FÜR
INFORMATIK
DER LUDWIG-MAXIMILIANS-UNIVERSITÄT MÜNCHEN

LS Prof. Kranzlmüller

Praktikum Rechnernetze

Kapitel 3: Autonome Systeme

MNM
TEAM
Munich Network Management Team

3 Das Internet und Autonome Systeme

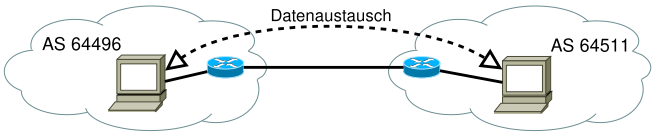
Inhaltsverzeichnis

3.1 Vermittlungsschicht (OSI-Schicht 3) 40
3.1.1 Fragmentierung 41
3.1.2 Tunneling 41
3.2 Internet Protocol Version 6 42
3.2.1 Adressen 43
3.2.2 Hilfsprotokolle 45
3.2.3 Koexistenz von IPv4 und IPv6 45
3.3 Routing-Verfahren 46
3.3.1 Optimale Pfade 46
3.3.2 Distanz-Vektor Verfahren 47
3.3.3 Link-State Verfahren 47
3.3.4 Inter-AS-Routing 48
3.3.5 FRRouting Suite 49
3.4 Aufgaben 52

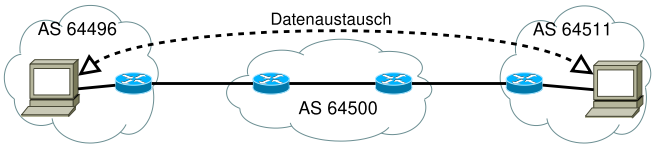
Das Internet ist ein Verbundnetz, bestehend aus den Netzen vieler verschiedener Betreiber (Unternehmen, Bildungseinrichtungen, Militär, usw), die jeweils eigene, administrativ unabhängige Netze betreiben. Das Verbinden zweier Netze verschiedener Netzbetreiber geschieht meist aus gegenseitigem Nutzen (Rechner in dem einen Netz können mit Rechnern im anderen Netz kommunizieren), oder wirtschaftlichem Interesse. In diesem Zusammenhang nennt man die unabhänigigen administrativen Domänen eines Betreibers *Autonome Systeme* (AS). Ein AS ist genehmigungspflichtig (bei der Internet Assigned Numbers Authority, IANA) und trägt eine weltweit eindeutige *AS-Nummer* – diese stellt eine Art Adresse der administrativen Domäne dar.

Grundsätzlich wird unterschieden, ob es sich bei der Verbindung zweier Netze bzw. AS um eine sogenannte *Peering*-Beziehung handelt, oder um eine *Transit*-Beziehung. Abbildung 3.1(a) zeigt AS-Peering, das den Netzteilnehmern in den Netzen der Peering-Partner Kommunikation *in das Netz* des anderen Peering-Partners erlaubt. Transit bedeutet die Erlaubnis, Nachrichten *durch das Netz* eines Betreibers zu vermitteln, auch wenn sie an Endsysteme außerhalb dessen Netz adressiert sind: Das Netz, durch das vermittelt wird dient also als „Durchfahrtstraße". In Abbildung 3.1(b) benötigen die beiden äußeren AS ein Transitabkommen mit dem AS 64500, damit ein Datenaustausch zwischen den beiden Endgeräten erfolgen kann.

Analog zu privaten IP-Adressen wurden AS-Nummern für private Nutzung (z.B. im Testbetrieb) reserviert, sowie zur Nutzung in Dokumentation:



(a) Peering-Beziehung zwei benachbarter Autonomer Systeme



(b) Transit-Beziehungen zwischen benachbarten Autonomen Systemen

Abb. 3.1: Zwei Arten von Beziehungen zwischen Autonomen Systemen erlauben den Datenaustausch zwischen zwei Rechnern

```
64512-65534 Designated for private use (Allocated to the IANA)
64496-64511 Reserved for use in documentation and sample code  [RFC5398]
```

Die Versuche im Praktikum nutzen diese privaten AS-Nummern.

3.1 Vermittlungsschicht (OSI-Schicht 3)

Aufbauend auf der Sicherungsschicht (siehe Kapitel 2.2), die dafür zuständig ist Rahmen durch ein LAN zu transportieren, werden mit den Funktionen der Vermittlungsschicht Nachrichten von der Quelle, durch das Transportnetz, bis zum endgültigen Ziel übertragen. Dabei können Teilnetze durchquert werden, die beliebige Schicht 2 Implementierungen einsetzen. Dadurch wird es ermöglicht, Daten zwischen Endpunkten in heterogenen Netzen auszutauschen. Die Komponenten der Schichten 1, 2 und 3 bilden zusammen das Transportsystem.

Da die Vermittlungsschicht Datagramme bis zum endgültigen Ziel überträgt, müssen alle Endpunkte über alle Teilnetze hinweg eindeutig adressierbar sein. Mit dem geplanten Einsatzzweck viele Schicht 2 Netze beinhalten zu können, muss bei der Entwicklung von Schicht 3 Protokollen davon ausgegangen werden, dass man mit sehr vielen Teilnehmern zurecht kommen muss, im Gegensatz zu einem einzelnen Schicht 2 Teilnetz.

Zwei Hauptaufgaben der Schicht 3 sind also eine globale (netzweite), eindeutige Adressierung aller möglicher Endpunkte und die Verschattung der zugrunde liegenden Übertragungstechniken. Protokolle höherer Schichten müssen sich dadurch nicht mehr mit Anzahl, Art und Topologie der Komponenten des Transportnetzes befassen.

3.1.1 Fragmentierung

Oft wird die maximale Länge einer Schicht-N-PDU durch die darunter liegende Schicht beschränkt. Auf dem Weg zwischen Sender und Empfänger durchquert ein Datagramm (Schicht-3-PDU) häufig verschiedene Subnetze bzw. LANs, deren Schicht-2-PDUs unterschiedliche maximale Längen für Nutzdaten zulassen. Ein Datagramm gegebener Größe muss daher in kleinere *Fragmente* aufgeteilt werden, jedesmal wenn ein Subnetz passiert werden soll, dessen MTU (engl. Maximum Transmission Unit) kleiner ist, als die Größe des Datagramms. Ethernet (ohne Erweiterungen) erlaubt z.B. bis zu 1500 Bytes Nutzdaten pro Rahmen.

Fragmentierung von IP-Paketen

Wird ein IP-Paket fragmentiert, so sind die entstehenden Fragmente ebenfalls IP-Pakete. Im IP-Header eines jeden Fragments bestimmt das *Fragment Offset*-Feld die Position der Daten (in Einheiten von 64-Bit) im ursprünglichen IP-Paket. Das *more*-Flag zeigt an, dass auf ein Fragment ein weiteres Fragment folgt. Bis auf das letzte Fragment hat jedes Fragment das more-Flag im Header gesetzt. Mit Hilfe des Fragment Offset-Felds und dem more-Bit kann ein Empfänger der Fragmente diese wieder zusammensetzen und ggf. das ursprüngliche IP-Paket weiterleiten.

3.1.2 Tunneling

Tunneling ist eine weitere Methode um ein logisches Netz in einem vorhandenen Netz zu implementieren. Eine ähnliche Technik haben Sie mit VLANs im vorherigen Abschnitt kennengelernt. Im Gegensatz zu VLANs, die eine logische Topologie mit mehreren Endpunkten auf einen physischen Aufbau aufbringen, zielen Tunnel darauf ab, eine direkte (logische) Verbindung zwischen zwei Endpunkten zu erstellen. Dabei wird das Netz (oder auch mehrere Netze) zwischen den Endpunkten zu einer virtuellen Leitung abstrahiert.

Durch diese Technik können Teilnetze transparent zusammengeschaltet werden. Sie wird häufig eingesetzt, um entfernte Standorte in ein internes Firmennetz zu integrieren oder mobilen Endgeräten mit Internetzugang Zugriff auf die internen Ressourcen zu ermöglichen.

Abbildung 3.2 zeigt ein IP-Paket, das vom Rechner „Sender" an „Empfänger" gesendet wird. Bis zum ersten Router A wird das IP-Paket als Nutzdaten, beispielsweise innerhalb eines Ethernet-Rahmens, transportiert. Router A behandelt das IP-Paket wie Nutzdaten einer höheren Schicht: er erstellt ein äußeres IP-Paket, schreibt unser ursprüngliches Paket in das Nutzdatenfeld des äußeren IP-Pakets und vermittelt das äußere Paket weiter auf dem Pfad zu Router B. Dieser stellt das ursprüngliche IP-Paket dem Empfänger zu. Das Netz zwischen den Routern A und B wird dabei aus Sicht unseres (inneren) IP-Pakets zu einer direkten Verbindung – einem Tunnel – abstrahiert.

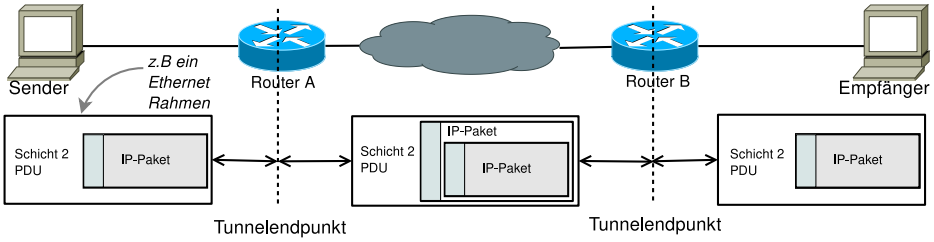


Abb. 3.2: Ein IP-Paket wird für den Transit zwischen den Routern A und B in ein weiteres IP-Paket gekapselt

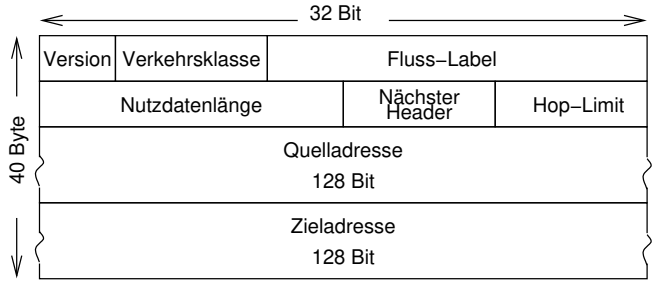


Abb. 3.3: Aufbau des IPv6 Headers

3.2 Internet Protocol Version 6

Seit der Einführung von IPv4 in den 1980er Jahren haben sich die Anforderungen an das Protokoll verändert. Dies spiegelt sich in nachträglichen Änderungen und Anpassungen wieder. Zum Beispiel wurde die Bildung von Subnetzen Jahre nach der Veröffentlichung von [RFC 791] (IPv4) mit [RFC 1519] (CIDR) verändert. Durch das stetige Wachstum der Anzahl an Endgeräte im Internet und die hierarchische Aufteilung von IP-Adressen beinhaltet der 32-Bit Adressraum von IPv4 nicht genug Adressen für den Bedarf im Internet. Der 128-Bit Adressraum von IPv6 ist ein wesentlicher Grund für dessen Entwicklung als designierter Nachfolger von IPv4.

Als eigenständige Neuentwicklung konnte IPv6 ([RFC 2460]) von den Erfahrungen mit IPv4 profitieren. So wurde IPv6 mit vielen Funktionen und Eigenschaften entwickelt, mit der Absicht Probleme von IPv4 zu lösen und die Erweiterung des Protokolls zu erleichtern. Einige der wichtigsten Eigenschaften von IPv6 sind:

128-Bit Adressen IPv6-Adressen sind 128-Bit lang und werden hierarchisch in Subnetze aufgeteilt. Eine IPv6-Adresse setzt sich aus einer 64-Bit langen Subnetz-ID und einer 64-Bit langen Host-ID zusammen. Die größe des Adressraums für Host-IDs erlaubt es (weltweit) eindeutige Host-IDs zu vergeben. Dafür sieht die ursprüngliche IPv6-Spezifikation einen entsprechenden Algorithmus zur automatischen Generierung von Host-IDs vor. Das Ziel war die eindeutige Identifizierung

eines Hosts unabhängig von der Subnetz-ID und damit von dem Teilnetz indem sich ein Host befindet zu machen.

Besonders für mobile Endgeräte ist dies ein relevanter Aspekt, denn diese können häufig von einem (Funk-)Netz in ein anderes wechseln. Unter Anderem ist es hier für die Abrechnung (siehe Abschnitt „Abrechnungs-Management") wichtig mobile Endgeräte über Subnetze hinweg eindeutig identifizieren zu können. Heute ist man von dem Gedanken der eindeutigen Adressierung zum Schutz der Privatsphäre der Benutzer abgekommen, allerdings ist die technische Möglichkeit nach wie vor gegeben.

Feste Header-Länge und Header-Extensions Für die Vermittlung einer Nachricht ist es wichtig die Länge des IP-Headers zu kennen, da dieser eventuell relevante Informationen für die Vermittlung auf Schicht 3 enthält. ☺ Der IPv6-Ansatz sieht einen Header fester Länge vor, der durch *Header-Extensions* um zusätzliche Informationen erweitert werden kann. Zu diesem Zweck ist im IPv6-Header das Feld „Next-Header" vorgesehen (siehe Abbildung 3.3). Dies ermöglicht es den IPv6-Header nur bei Bedarf um Funktionen zu erweitern. Außerdem kann IPv6 zu einem späteren Zeitpunkt leicht durch das Definieren neuer Header-Extensions erweitert werden. Das erste Feld einer jeden Header-Extension ist ein Next-Header-Feld. Auf diese Art und Weise können Header-Extensions kombiniert werden.

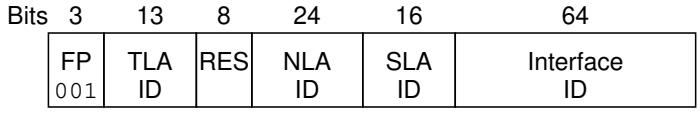
IPsec wurde ursprünglich als [RFC 2401] eingeführt, um bereits auf der Ebene der Vermittlungsschicht Daten verifizieren und verschlüsseln zu können. IPsec wurde komplett in IPv6 übernommen und kann mittels Header-Extensions eingesetzt werden.

Autoconfiguration Unter dem Stichwort *Autoconfiguration* werden mehrere Funktionen und Eigenschaften von IPv6 zugefasst. Ein prominenter Vertreter aus dieser Kategorie ist das automatische Generieren der Host-ID, zum Beispiel (aber nicht unbedingt!) aus der MAC-Adresse. Des weiteren implementiert IPv6 Gültigkeitsbereiche (engl. Scopes) von IP-Adressen. Die hier relevanten Scopes sind „link-local" und „global". IP-Adressen deren Scope link-local ist werden nicht geroutet. Aus einem im Standard definierten Präfix und der automatisch generierten Host-ID erhält jede Schnittstelle, auf der IPv6 eingesetzt wird, eine eindeutige link-local IP-Adresse.

3.2.1 Adressen

IPv6-Adressen werden als Folge von 16 Bit langen Segmenten notiert. Jedes der acht Segmente wird als Hexadezimalzahl aufgeschrieben, wobei je zwei Segmente (Oktettenpaare) durch einen Doppelpunkt ':' getrennt werden, zum Beispiel die Adresse `1080:0:0:0:8:800:200C:417A`.

Aufgrund des großen Adressraums kommen oft Bereiche einer Adresse vor, in denen mehrere aufeinanderfolgende Oktettenpaare den Wert null haben. Zugunsten einer verkürzten Schreibweise darf (höchstens einmal) eine Folge von Nullen zu zwei aufeinander



Feld	Name
FP	Format Prefix
TLA ID	Top-Level Aggregation Identifier
RES	reserviert
NLA ID	Next-Level Aggregation Identifier
SLA ID	Site-Level Identifier

Abb. 3.4: Struktur einer Global Unicast IPv6-Adresse

folgenden Doppelpunkten '::' verkürzt werden. Die Anzahl der Null-Oktette, die so repräsentiert werden kann offensichtlich aus der festen Länge einer Adresse und den ausdrücklich angegebenen Oktetten abgeleitet werden, z.B:

nicht verkürzt	`1080:0:0:0:8:800:200C:417A`
verkürzt	`1080::8:800:200C:417A`

Aufbau einer Adresse Die Abkehr von Netzklassen, wie sie ursprünglich in IPv4 verwendet wurden, eröffent neue Möglichkeiten für die Strukturierung von IP-Adressen, insbesondere für die Subnetz-ID. Die Struktur von IPv6-Adressen ist dahingehend gestaltet, den organisatorischen Gegebenheiten im Internet gerecht zu werden: es werden seitens verschiedener Organisationen Kern- und Transitnetze betrieben, sowie Zugangsnetze und lokale Netze. Dies wird abgebildet auf eine dreistufige Adresshierarchie. Abbildung 3.4 zeigt die Aufteilung einer *Aggregatable Global Unicast Address* in Segmente, die aus der Adresshierarchie abgeleitet werden. Die ersten 48 Bit der Adresse bezeichnen die **Public Topology** (insbesondere die Felder TLA und NLA), zum Beispiel Transitnetze. Die zweite Hierarchietufe ist die **Site Topology** (SLA-Feld), die standortbezogene Adressierung bezweckt. Die dritte Hierarchiestufe wird durch den **Interface Identifier** repräsentiert. Diese entspricht am ehesten der aus IPv4 bekannten Host-ID.

Spezielle Adressen Auch bei IPv6 sind bestimmte Adressen bzw. Präfixe mit besonderer Bedeutung belegt, unter anderem gibt es Festlegungen für eine nicht definierte Adresse für eine Schnittstelle, einer loopback-Adresse und privaten Adressen.

nicht definiert	`0:0:0:0:0:0:0:0` (bzw. '`::0`')
loopback	`0:0:0:0:0:0:0:1` (bzw. '`::1`')
private Adressen	Private Adressen sind solche, die nicht im Internet vermittelt werden. Dazu gehören bei IPv4 beispielsweise die Adressen des 10.*-Netzes. Bei IPv6 gehören dazu auch die sogenannten *link-local*-Adressen, mit dem 10-Bit-Präfix `1111 1110 10` bzw. das Subnetz `FE80::0/10`.

3.2.2 Hilfsprotokolle

Auch für IPv6 benötigt man bestimmte Funktionen, die in Hilfsprotokollen implementiert werden. Die bekannten Funktionen der (IPv4-)Hilfsprotokolle ARP und ICMP sind in ICMPv6 ([RFC 4443]) konsolidiert. Die Neuauflage war notwendig, da IPv6 subtile konzeptionelle Unterschiede zu IPv4 aufweist. Zum Beispiel geht IPv6 von Punkt-zu-Punkt Verbindungen auf Schicht 2 aus, während ARP für den Betrieb in einem Broadcast-Netz erforderlich ist. Infolgedessen ist z.B. die Auflösung IP-Adresse zu MAC-Adresse bei IPv6 deutlich komplexer.

Des weiteren implementiert ICMPv6 Funktionen aus dem Bereich Autoconfiguration, der erst mit IPv6 eingeführt wurde. Hier gibt es Überschneidungen mit den Funktionen von DHCP. Deshalb muss auch das DHCP-Protokoll angepasst werden um mit IPv6 zusammen zu arbeiten. Analog zu ICMP heißt die IPv6 Variante des DHCP-Protokolls DHCPv6 ([RFC 3315]). Zum Beispiel beinhaltet ICMPv6 „Router Advertisements", mit deren Hilfe Client-Rechner automatisch eine globale (geroutete) IPv6-Adresse und eine Default-Route erhalten. Dies sind typische Eigenschaften, die bei IPv4 nur über DHCP zugewiesen werden (können).

3.2.3 Koexistenz von IPv4 und IPv6

Seit geraumer Zeit wird die Umstellung des (gesamten) Internet auf die Version 6 von IP beschworen; tatsächlich findet diese Umstellung jedoch nur dort statt, wo ein akuter Bedarf an freien IP-Adressen (z.B. in asiatischen Ländern) besteht, oder bestimmte Eigenschaften von IPv6 zwingend erforderlich werden. Daher ist für die nächste Zeit eine Koexistenz der beiden Protokollversionen abzusehen.

Um eine Kompatibilität zwischen IP-Netzen und -Hosts zu gewähren, obwohl diese mit verschiedenen IP-Versionen arbeiten, existieren mehrere prinzipielle Ansätze:

- Dual-Stack: Knoten besitzen sowohl IPv4- als auch IPv6-Implementierung.
- Tunneling: Kapselung von Paketen im tunnelnden Protokoll, d.h. IPv6-Verkehr über IPv4-Netz bzw. IPv4-Verkehr über IPv6-Netz.
  Herausforderungen: Management des Tunnels (Erstellung, Abbau), Umgang mit Fragmenten

- Übersetzung von IPv4/IPv6-PDU in die jeweils andere Protokollversion. Diese Übersetzung kann in Übergangsknoten/Router geschehen, bzw. in Programmbibliotheken (APIs zur Netzprogrammierung) des Betriebssystems.

3.3 Routing-Verfahren

In Netzen mit vielen Routern und einer großen Anzahl von Verbindungen in andere Netze ist es sehr aufwändig Routing-Tabellen manuell zu verwalten. Zur Erleichterung setzt man *Routing-Protokolle* ein, die Router den Austausch von Informationen ermöglicht, insbesondere Routen. Indem Router „Wissen" über erreichbare Ziele austauschen können Routing-Tabellen automatisch erstellt und aktualisiert werden.

Adaptive Routing-Verfahren zeichnen sich dadurch aus, dass sie sich dem Zustand des Netzes anpassen. Durch den Einsatz von Routing-Protokollen erhält ein Router ereignis- oder zeitgesteuert aktuelle Informationen über die Erreichbarkeit entfernter Netzsegmente. Router nutzen diese Informationen um die eigene Routing-Tabelle zu ergänzen oder zu modifizieren. Dadurch „lernen" Router Pfade in entfernte Netzsegmente und können auf Veränderungen in der Topologie reagieren. Unter den adaptiven Routing-Verfahren wird zwischen *zentralen* und *verteilten* Verfahren unterschieden. Bei zentralen Verfahren erhalten Router Informationen von fest definierten Stationen, während bei verteilten Routing-Verfahren die Router als gleichgestellte Einheiten (engl. peers) interagieren. Die im folgenden dargestellten Verfahren sind *verteilte* Verfahren und werden daher auf jedem Router ausgeführt. Heute verwenden die meisten Routing-Protokolle das adaptive *Distanz-Vektor Verfahren* oder das adaptive *Link-State Verfahren*.

3.3.1 Optimale Pfade

Die Hauptfunktion von Routing-Protokollen besteht darin, dass jeder Router Information über die ihm bekannten Subnetze anderen Routern mitteilt, so dass Rechner aus unterschiedlichen Subnetzen interagieren können. Dabei ist eine wichtige Herausforderung die Ermittlung eines *optimalen Pfades* zu jedem Subnetz. Mit einer Kostenfunktion werden die Transitkosten für jede Verbindung zwischen zwei benachbarten Routern festgelegt. Auf dem resultierenden Graphen mit gewichteten Kanten (wobei die Knoten des Graphen Router im Netz sind und die Kanten Leitungen zwischen den Routern) sucht ein Planungsalgorithmus einen optimalen Pfad zu jedem Subnetz.

Kostenfunktionen dienen dazu den Planungsalgorithmus zu beeinflussen. Eine Nachricht die sich im Transitnetz befindet belegt Ressourcen in Routern und Switches. Da diese Ressourcen meist sehr begrenzt sind, wird die Kostenfunktion häufig so gewählt, dass eine Nachricht so schnell wie möglich das Transitnetz verlässt. Eine Nachricht verlässt das Transitnetz entweder durch direkte Zustellung an den Adressaten oder indem sie in ein Netz eines anderen Betreibers weitergeleitet wird. Beispiele für Kostenfunktionen sind „Anzahl Hops", d.h. die Anzahl der Zwischenschritte, oder die summierte Latenz der Leitungen auf einem Pfad.

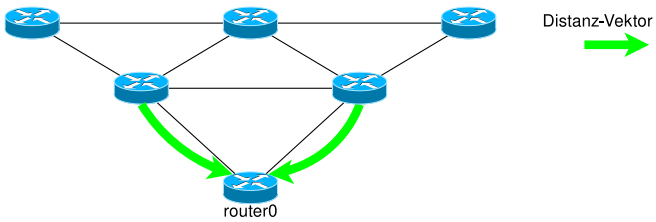


Abb. 3.5: Beim Distanz-Vektor Verfahren erhält router0 nur von den direkt benachbarten Routern Informationen

3.3.2 Distanz-Vektor Verfahren

Beim Distanz-Vektor Verfahren (DV) erstellt jeder Router eine Liste aller erreichbaren Ziele und die zugehörigen Kosten für den Transit. Indem benachbarte Router ihre Listen austauschen, „lernen" sie Pfade zu allen Zielen, die über ihre Nachbarn erreicht werden können. Der zugrunde liegende Algorithmus für das Distanz-Vektor Verfahren ist der Bellman-Ford oder Ford-Fulkerson Algorithmus.

Ein Router kennt und interagiert nur mit seinen direkten Nachbarn. Ein Ziel wird entweder direkt, oder über einen der Nachbarn erreicht. Die für das Verfahren benötigten Informationen über erreichbare Ziele erhält ein Router aus seiner Routing-Tabelle. Die Kosten werden über eine Kostenfunktion ermittelt. Ein einfaches Beispiel einer Kostenfunktion für das Distanz-Vektor Verfahren ist die Anzahl der Zwischenschritte (oder Entfernung, engl. distance), mit der ein Rechner im Zielnetz erreicht werden kann. Direkt erreichbare Netze erhalten einen Wert von 0 für die „Kosten" der Erreichbarkeit, da keine Zwischenschritte benötigt werden. Die Kosten der indirekt erreichbaren Ziele entspricht der länge des Pfades (Anzahl vermittelnder Router) zum Zielnetz.

Die vollständige Liste mit *Routing-Informationen* wird an alle direkten Nachbarn geschickt. Empfängt ein Router Routing-Informationen, vergleicht er deren Inhalt mit der eigenen Routing-Tabelle und den darin gespeicherten Kosten. Ziele, die noch keinen Eintrag in der eigenen Routing-Tabelle haben, werden ergänzt. Da die Informationen von einem direkten Nachbarn stammen, werden die Kosten (Distanz) um eins erhöht, bevor ein Eintrag in die eigene Routing-Tabelle eingefügt wird. Existiert bereits ein Eintrag in der Routing-Tabelle entscheidet der Router anhand der Kosten ob der Eintrag in der Routing-Tabelle ersetzt oder beibehalten wird. Bei dieser Entscheidung wird stets der Pfad mit weniger Zwischenschritten bevorzugt. Abbildung 3.5 zeigt ein Netz aus Routern, die ein Distanz-Vektor Verfahren einsetzen. In diesem Beispiel schicken nur die beiden direkt benachbarten Router Informationen an router0.

3.3.3 Link-State Verfahren

Ein Router, der das Link-State Verfahren einsetzt, berechnet optimale Pfade zu jedem Ziel, z.B. mit dem Dijkstra Algorithmus. Dazu ist es notwendig, dass jeder Router die gesamte Topologie kennt.

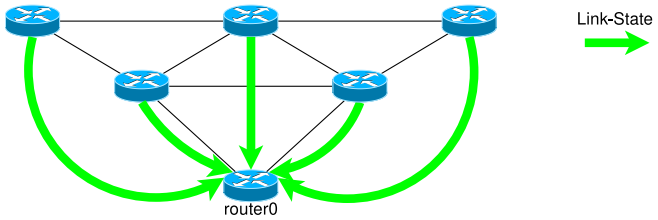


Abb. 3.6: Beim Link-State Verfahren erhält router0 von allen teilnehmenden Routern Informationen

Teilnehmende Router erstellen eine Liste mit allen direkt erreichbaren Zielen und mit allen benachbarten Routern. Mit einer Kostenfunktion werden die Kosten für den Transit zu benachbarten Routern bestimmt. Ein Beispiel für eine Kostenfunktion ist die *Round Trip Time* zwischen den Routern. Für die Vermittlung zu den Zielen werden keine Kosten festgelegt. Die gesammelten Informationen werden an alle anderen Router verteilt. Die hierzu ausgetauschten Nachrichten werden *link state advertisements* (LSA) genannt. Aus den LSAs erstellt jeder Router einen Graphen mit gewichteten Kanten und berechnet für sich die günstigsten Pfade zu allen Zielen.

Abbildung 3.6 zeigt ein Netz aus Routern, die ein Link-State Verfahren einsetzen. In diesem Beispiel schicken alle teilnehmenden Router Informationen an router0. Link-State Verfahren benötigen im Vergleich mit Distanz-Vektor Verfahren deutlich mehr Interaktion zwischen den Routern und Ressourcen zur Berechnung der optimalen Pfade. Mit diesen Nachteilen erhält man die Vorteile, dass Pfadberechnungen mit allen Informationen und unabhängig voneinander erfolgen. Dadurch sind mit Link-State Verfahren berechnete Pfade immer optimal, während beim Distanz-Vektor Verfahren Pfade iterativ, durch den Austausch von Routing-Informationen, zu optimalen Pfaden *konvergieren*.

3.3.4 Inter-AS-Routing

Zwischen Autonomen Systemen werden Wegewahlverfahren eingesetzt, die politische oder wirtschaftliche Interessen der Betreiber berücksichtigen. Standardmäßig wird dabei das Border Gateway Protocol ([RFC 4271]) eingesetzt, das mittels eines sogenannten Pfadvektor-Verfahrens (ähnlich dem Distanz-Vektor-Verfahren) Datenverkehr zwischen autonomen Systemen leitet.

Routing-Protokolle, die zwischen autonomen Systemen eingesetzt werden nennt man EGP (Exterior Gateway Protocol, BGP ist ein wichtiger Vertreter für EGP); sie ermitteln Pfade zwischen sogenannten Border Routern. Border Router unterscheiden sich dahingehend von „normalen" Routern, dass Border Router über mindestens eine logische Verbindung in ein anderes AS verfügen. Andere Router sind lediglich für die Vermittlung innerhalb eines AS zuständig.

Kosten-„Metriken" bei EGP sind meist monetär, z.B. EUR/Datenvolumen oder aber durch Regeln mit vertraglichem Hintergrund vorgegeben. Routing-Protokolle, die innerhalb

AS benutzt werden, nennt man IGP (Interior Gateway Protocol, z.B. RIP, OSPF). Die Border Router geben Informationen, die sie durch ein EGP gelernt haben mittels IGP an die inneren Router weiter. Ob und welche Routen wirklich zwischen EGP und IGP ausgetauscht werden bleibt der Konfiguration der Router überlassen.

Durch die Unterscheidung zwischen EGP und IGP entsteht eine Routing-Hierarchie. Während IGPs meist darauf ausgerichtet sind kürzeste Wege zu wählen, berücksichtigen EGPs andere, nicht-technische Aspekte.

3.3.5 FRRouting Suite

FRRouting (FRR) ist ein GPL-lizenziertes Software-Paket, das Implementierungen verschiedener Routingprotokolle, wie z.B. RIP, OSPF und BGP bereitstellt. Die Kernkomponente ist der *zebra*-Dienst, der als Schnittstelle zum Betriebssystem dient. Für jedes in FRR enthaltene Routing-Protokoll existiert ein eigenes Programm (daemon) etwa `ripd` für RIP, `ospfd` für OSPF usw.

Jeder Daemon bzw. Dienst kann über eine Kommandozeile (genannt „vty") administriert werden, die sich an der Bedienung anderer Routing-Programme orientiert. Damit nicht jeder Dienst separat verwaltet werden muss, existiert zusätzlich das Werkzeug "vtysh", über das alle Prozesse gemeinsam administriert werden können.

In der Datei /etc/frr/daemons konfiguriert man welche Dienste beim Start von FRR aktiviert werden.

```
zebra=yes
bgpd=no
ospfd=yes
ospf6d=no
ripd=no
ripngd=no
```

Um FRR und die damit verknüpften Dienste zu starten oder anzuhalten verwendet man den Befehl:

```
# /etc/init.d/frr <AKTION>
```

```
FRR starten:                          FRR anhalten:
# /etc/init.d/frr start               # /etc/init.d/frr stop
```

Eine ausführliche Dokumentation der FRRouting Suite, über die darin enthaltenen Dienste und deren Konfiguration, finden Sie unter [WWW 18].

vtysh

Nach dem Aufruf von `vtysh` befindet man sich auf der gemeinsamen Konsole aller aktivierten Routing-Dienste. Dort kann man sich jederzeit durch die Eingabe von `?` eine Liste der möglichen Befehle anzeigen lassen:

```
vtysh# ?
  enable   Turn on privileged mode command
  exit     Exit current mode and down to previous mode
  list     Print command list
```

```
  ping     Send echo messages
  quit     Exit current mode and down to previous mode
  show     Show running system information
  ...
```

Dies ist insbesondere bei teilweise eingegebenen Befehlen hilfreich. Wenn man bereits einen Teil des Befehls eingegeben hat und mit einem `?` (Fragezeichen) abschließt, werden alle weiteren Optionen angezeigt, die statt dem `?` stehen können:

```
vtysh# show ip ?
  ...
  bgp      BGP information
  ospf     OSPF information
  rip      Show RIP routes
  ...
```

Es gibt verschiedene VTY Modi. Der *View-Modus* erlaubt nur lesenden Zugriff auf die Konfiguration des Routers, während der *Enable-Modus* sowohl lesenden als auch schreibenden Zugriff erlaubt. Nach dem Aufruf von `vtysh` befindet man sich im View-Modus. Mit dem Kommando `enable` wechselt man in den Enable-Modus. Mit `disable` kehrt man zurück in den View-Modus. Möchten Sie die Konfiguration anpassen, so müssen Sie zunächst in den *Configuration Mode* wechseln.

In den *Configuration Mode* gelangen Sie durch den Aufruf von `configure terminal`. Nun stehen Ihnen weitere Befehle zur Verfügung. Beispielsweise können Sie jetzt mit `interface eth1` in den Konfigurationsmodus für das Interface eth1 wechseln und dort wiederum z.B. die IP-Adresse ändern. Mit dem Befehl `exit` verlassen Sie den aktuellen Modus und kehren zum vorherigen zurück. Die Wirkung von Befehlen, die eine Funktion aktivieren oder deaktivieren, können Sie durch Voranstellen von `no` umkehren.

Die gesamte Abfolge um z.B. die IP-Adresse von Interface eth1 zu ändern sieht so aus:

```
(1) router1# vtysh
(2) router1-vtysh# enable
(3) router1-vtysh# configure terminal
(4) router1-vtysh(config)# interface eth1
(5) router1-vtysh(config-if)# ip address 10.0.0.1/8
(6) router1-vtysh(config-if)# exit
(7) router1-vtysh(config)# exit
(8) router1-vtysh# exit
```

Erklärung: (1) `vtysh` starten, (2) Enable Mode aktivieren (nur nötig, falls noch nicht aktiviert), (3) Konfigurationsmodus aktivieren, (4) Konfigurationsmodus von Interface eth1 aufrufen, (5) IP-Adresse ändern, (6) Interface-Konfigurationsmodus verlassen, (7) Konfigurationsmodus deaktivierten, (8) `vtysh` beenden

Neben der interaktiven Konfiguration über `vtysh` können sämtliche Einstellungen auch in Konfigurationsdateien verwaltet werden. Diese befinden sich standardmäßig im Verzeichnis `/etc/frr/` und werden mit `<daemonname>.conf` benannt, also z.B. `ripd.conf`. Wenn Sie FRRouting über `vtysh` konfiguriert haben, können Sie mit dem Befehl `write file` die entsprechenden Konfigurationsdateien automatisch erstellen lassen.

ripd

Die meisten Befehle zur Konfiguration des `ripd` erreichen Sie vom Konfigurationsmodus aus durch den Befehl `router rip`. Als minimale Konfiguration wird nur eine Angabe benötigt, auf welchen Interfaces RIP aktiviert werden soll. Dies geschieht durch den Befehl:
`network <IFNAME>|<IP-PREFIX>`
Bei diesem Befehl geben Sie entweder ein Interface (z.B. `eth1`) oder ein IP-Prefix (z.B. `10.6.0.0/16`) an. Letzteres bezieht alle Interfaces mit ein, die sich im angegebenen Adressbereich befinden.

ospfd

Die Basiskonfiguration des `ospfd` verläuft analog zum `ripd`. Die Befehle lauten hier `router ospf` und `network <IP-PREFIX> area <AREA>`. Bei `<AREA>` reicht es für unsere Zwecke, wenn Sie `0` angeben.

bgpd

Um den `bgpd` zu administrieren, geben Sie im Konfigurationsmodus den Befehl `router bgp <AS-NUMMER>` ein, wobei Sie als AS-Nummer die Nummer Ihres autonomen Systems ersetzen. Nun können Sie mit dem Befehl `neighbor <IP-ADRESSE> remote-as <AS-NUMMER>` Border-Router anderer autonomer Systeme als Nachbarn hinzufügen. Zusätzlich sollten Sie mit dem Befehl `network <IP-PREFIX>` angeben, für welches Netz Ihr Router als Border-Router fungiert. (Der `network` Befehl hat hier also eine andere Semantik!) Mit `neighbor <IP-ADRESSE> weight <GEWICHT>` können Sie einem Nachbarn ein Gewicht zuordnen, wobei höher gewichtete Nachbarn beim Routing bevorzugt werden.

3.4 Aufgaben

A300 Fragmentierung und IP-Tunneling (Theorie)

i) In wie viele Fragmente wird ein IP-Paket mit Größe 9000 Byte zerlegt, um über ein Ethernet mit MTU = 1500 Byte übertragen zu werden? Geben Sie eine vollständige Rechnung an!

ii) Nennen Sie drei Gründe dafür, dass Netzbetreiber IP-Fragmentierung in ihren Netzen verbieten. Erläutern Sie diese Gründe!

iii) Wie wird in modernen TCP/IP-Implementierungen dafür gesorgt, dass Fragmentierung in der Regel nicht erforderlich ist?

A301 IPv6 (Theorie)

i) Erklären Sie anhand eines Beispiels auf einer Ihrer VMs wie link-local Adressen aus der MAC-Adresse abgeleitet werden!

ii) Wie implementiert IPv6 „Broadcasts"?

iii) Welches sind die privaten Adressbereiche in IPv6, analog zu 10.0.0.0/8, 172.16.0.0/12 und 192.168.0.0/16 in IPv4?

iv) Für besondere Zwecke, außer für den privaten Gebrauch, sind noch weitere Bereiche reserviert. Wie teilt sich der IPv6 Addressraum auf? *Hinweis:* IANA, ignorieren Sie die vom IETF reservierten Bereiche

A302 Fragmentierung und Tunneling

i) Legen Sie den Pfad (pc1, router1, router4, router3, router2, pc2) an, so dass Nachrichten zwischen pc1 und pc2 vermittelt werden und zeigen Sie mittels `traceroute`, dass die Nachrichten entlang dieses Pfades vermittelt werden!

ii) Bestimmen Sie mittels `ip` die MTU der genutzten Schnittstellen! Erstellen Sie für die Ausarbeitung eine Tabelle mit den Spalten: Rechnername, Schnittstelle, IP-Adresse, MTU.

iii) Beschränken Sie nun mittels `ip` die MTU von router3.eth3 auf 1000 Bytes und die MTU von router4.eth3 auf 1100 Bytes!

Beschreiben Sie das Verhalten der Rechner, wenn Sie IP-Nachrichten mit 1200 Bytes Nutzdaten von pc1 an pc2 schicken! Nutzen sie `ping` um Nachrichten dieser Größe zu erzeugen. Achten Sie darauf, dass `ping` das *Don't Fragment* Flag setzt.

iv) Wiederholen Sie den `ping`-Versuch ohne gesetztes *Don't Fragment* Flag. Wie verändert sich der Datenfluss im Vergleich zum vorherigen Versuch?

v) Weisen Sie dem Interface router3.eth3 eine beliebige IPv6 Adresse zu. Ist der Vorgang erfolgreich? Wenn nein, was ist die Ursache? Ziehen sie hierzu auch die beschriebenen Rahmenbedingungen aus [RFC 2460] in Betracht.

vi) Konfigurieren Sie einen IP-Tunnel zwischen router1 und router2 und konfigurieren Sie die Systeme so, dass sämtliche IP-Nachrichten zwischen pc1 und pc2 durch den Tunnel übertragen werden!

Hinweis: Benutzen Sie einen GRE-Tunnel, den Sie mit dem Programm `ip` einrichten. Hintergrundinformation zu GRE (Generic Routing Encapsulation) lesen Sie bitte in [RFC 2784] vor Durchführung des Versuchs nach; zur praktischen Nutzung finden Sie die Dokumentation in der Manpage `ip-tunnel (8)`.

vii) Zeigen Sie mit `traceroute`, dass alle Nachrichten durch den Tunnel übertragen werden!

viii) Vergleichen Sie die von `traceroute` angegebenen Zwischenstationen aus den vorangegangenen Versuchen. Nennen Sie alle Unterschiede!

ix) (Freiwillig) Fangen Sie mit Hilfe von `tcpdump` ein Paket ab, das den Tunnel gerade „betritt" bzw. „verläßt" und analysieren Sie die gekapselte Headerstruktur. Welche Unterschiede zwischen den Header-Daten des inneren versus denen des äußeren IP-Paketes stellen Sie fest?

Hinweis: Beim Arbeiten mit `tcpdump` kann es nötig sein den Detailgrad der Ausgabe (`-v`) oder die Mitschnitt-Größe der Pakete (`-s`) anzupassen. Für eine komfortablere Auswertung kann man zunächst die Rohdaten in eine Datei umleiten (`-w`). In diesem Fall erzeugt `tcpdump` keine Ausgaben mehr. Die Datei mit Rohdaten können Sie anschließend mit Wireshark auswerten.

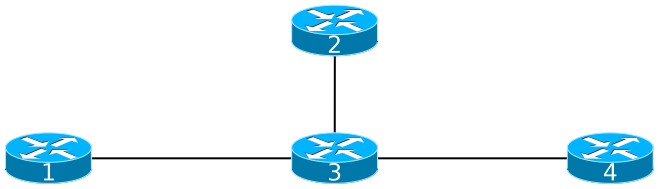


Abb. 3.7: Router 1 bis 4 mit kreisfreier Sterntopolgie

A303 IPv6

i) Entfernen Sie alle IPv4-Adressen von den Schnittstellen eth{1,2,3,4} auf allen Routern!

ii) Bilden Sie das Netz aus Abblidung 3.7 nach! Verwenden Sie DHCPv6-PD (prefix delegation) um ihr Netz hierarchisch aufzuteilen mit router3 als Zentrum. (*Hinweis:* `/etc/config/{network,dhcp}`, siehe OpenWrt-Dokumentation[1]) Beachten Sie, dass ...

   1) Sie IPv6-Adressen aus den Bereichen 2001:db8:⟨Gruppennummer⟩::/48 und fc00:dead:beef:⟨Gruppennummer⟩::/52 verwenden! (Die ersten 48 Bits sind gesetzt, Bits 48 bis 52 ersetzen Sie durch Ihre Gruppennummer, die restlichen Bits stehen zu Ihrer freien Verfügung.)
   2) Die Router IPv6-Pakete zwischen ihren Schnittstellen vermitteln!
   3) `ping`-Anfragen zwischen allen Routern vermittelt werden! Fügen Sie die Routing-Tabellen Ihrer Router in Ihre Ausarbeitung ein.

iii) Konfigurieren Sie Ihre Router so, dass die PCs automatisch IPv6-Adressen mit Ihrem Präfix erhalten. Erläutern Sie anhand von geeigneten Aufzeichnungen den Vorgang wie pc1 und pc2 eine globale IPv6-Adresse erhalten! Betrachten Sie auch die Kommunikation zwischen router3 und router{1,2}.

iv) Passen Sie die Routing-Konfigurationen von router1, router2 und router3 an, so dass IPv6-Pakete zwischen pc1 und pc2 vermittelt werden. Sind manuelle Änderungen daran notwendig? Zeigen Sie mittels `traceroute6` (8) den Pfad von IPv6-Paketen zwischen pc1 und pc2!

v) Konfigurieren Sie einen ip4ip6-Tunnel zwischen router1 und router2, so dass IPv4-Pakete von pc1 an pc2 (und zurück!) per IPv6 zwischen den Routern vermittelt werden! Zeigen Sie, dass Ihre Konfiguration funktioniert und IPv4-Pakete tatsächlich in IPv6 gekapselt werden.

[1] `https://openwrt.org/docs/guide-user/network/ipv6/configuration`

A304 Distanz-Vektor Routing mit RIP

Installieren Sie die FRRouting-Suite auf router1 bis 4. (*Hinweis:* Auf Debian: frr, auf OpenWrt: frr frr-zebra frr-watchfrr frr-staticd frr-vtysh frr-ripd frr-ospfd frr-bgpd)

i) Vernetzen Sie die Router wie folgt: router1–router2–router4–router3, deaktivieren Sie alle anderen Verbindungen zwischen den Routern! Vergeben Sie passende Subnetze und IP-Adressen aus Ihrem Gruppennetz 10.⟨*Gruppennummer*⟩.0.0/16, legen Sie keine manuellen Routen an!

ii) Erstellen Sie die Konfigurationsdatei `/etc/frr/ripd.conf`! Die Syntax entnehmen Sie der Dokumentation [2]. *Hinweis:* Auf Debian können Sie auch Beispielkonfigurationen in `/usr/share/doc/frr/examples` finden. Zeilen, die mit `!` anfangen, sind Kommentare.

iii) Passen Sie die Datei `/etc/frr/daemons` an und starten Sie die Dienste `zebra` und `ripd` nacheinander auf Ihren Routern! (`/etc/init.d/frr start`) *Hinweis:* starten Sie zuerst beide Dienste auf router1, dann auf router2, usw. usf.)

iv) Beobachten Sie auf router1.eth2 die ein- und ausgehenden RIP-PDUs! Zeigen Sie die Zwischenschritte in denen router1 sein Wissen über die Infrastruktur vervollständigt!

v) Machen Sie sich mit `vtysh` vertraut. Benutzen Sie `vtysh` um sich die Routing-Tabelle für router1 anzeigen zu lassen! Welche Bedeutung hat das Feld „From"?

vi) Aktivieren Sie die Verbindung router1-router3 und zeigen Sie die Zwischenschritte, bis sich die Routingtabellen von router2 und router4 an den neuen Zustand angepasst haben!

vii) Deaktivieren Sie router1.eth1, um einen Verbindungsausfall zu simulieren! Beobachten Sie das Verhalten der Router!

   a) Welches Verhalten der Router ist nach [RFC 1058] zu erwarten, um das über router1.eth1 erreichbare Subnetz aus den Routing-Tabellen zu entfernen?

   b) Welches Verhalten der Router ist tatsächlich zu beobachten? Welche Technik wird eingesetzt um das nun nicht mehr erreichbare Subnetz aus den Routing-Tabellen zu entfernen? Zeigen Sie die entsprechenden PDUs der Router!

[2] `https://docs.frrouting.org/en/latest/ripd.html`

A305 Link-State Routing mit OSPF

Im Folgenden wiederholen Sie den Versuch von oben. Anstatt RIP wird diesmal OSPF eingesetzt.

i) Stellen Sie den Zustand aus A304, i) wieder her. Deaktivieren Sie FRR auf allen Routern!

ii) Erstellen Sie die Konfigurationsdatei `/etc/frr/ospfd.conf`! Die Syntax entnehmen Sie der Dokumentation [3].

iii) Passen Sie die Datei `/etc/frr/daemons` an und starten Sie `zebra` und `ospfd` nacheinander! *Hinweis:* zuerst beide Dienste auf router1, dann router2, usw. usf.

iv) Beobachten Sie die ein- und ausgehenden OSPF-PDUs auf router1.eth2! Beschreiben Sie die Zwischenschritte, in denen router1 sein Wissen über die Infrastruktur vervollständigt!

v) Aktivieren Sie die Verbindung router1-router3. Beschreiben Sie die Zwischenschritte, bis sich die Routingtabellen von router2 und router4 an den neuen Zustand angepasst haben! Belegen Sie Ihre Erklärung mit entsprechenden Programmausgaben!

vi) Deaktivieren Sie router1.eth1, um einen Verbindungsausfall zu simulieren! Erläutern Sie den Unterschied zwischen RIP und OSPF in der Art und Weise wie der Verbindungsausfall an andere Router propagiert wird!

[3] `https://docs.frrouting.org/en/latest/ospfd.html`

A306 Autonome Systeme und BGP

Neben den Interior Gateway Protokollen wollen wir uns BGP als Exterior Gateway Protokoll ansehen. Behalten Sie Ihre OSPF-Konfiguration aus der vorherigen Aufgabe bei.

i) Stellen Sie sicher, dass Sie Adressen aus Ihrem Subnet (10.⟨*Gruppe*⟩.0.0/16) verwenden. Ihre Infrastruktur bildet das Autonome System mit der Nummer 65000 + ⟨*Gruppe*⟩.

ii) Für diese Aufgabe werden vom Lehrstuhl zwei weitere Maschinen bereitgestellt, Gruppe11 (Odd, AS 65011) und Gruppe12 (Even, AS 65012). Auf diesen Maschinen ist BGP bereits konfiguriert. Abhängig von ihrer Gruppennummer können Sie allerdings nur mit einer der Maschinen direkt kommunizieren. Ihr Ziel ist es mittels BGP eine Route in das Subnet der anderen Maschine zu bekommen.

   Stellen Sie mit Hilfe eines GRE-Tunnels eine Verbindung zu der Maschine her. Verwenden Sie als Endpunkte die Adresse von router1 an eth0 und die globale IP-Adresse der Zielmaschine. Achten Sie darauf, dass eine TTL > 3 verwendet wird. *Hinweis:* Sie können auf die selbe Weise eine Verbindung zu anderen Gruppen herstellen.

iii) Konfigurieren Sie Router1 als Border Router Ihres Autonomen Systems. Vermitteln Sie mit dem Border Router am anderen Ende des Tunnels. Dieser hat die IP-Adresse 10.11.0.⟨*Ihre Gruppe*⟩ bzw. 10.12.0.⟨*Ihre Gruppe*⟩. *Hinweis:* Definieren Sie Host-Routen.

iv) Passen Sie die OSPF-Konfiguration von Ihren Routern so an, dass die über BGP gelernten Routen auch den anderen Routern im jeweiligen Netz mitgeteilt werden. *Hinweis:* Sie können dazu entweder mit `vtysh` arbeiten, oder die Datei `/etc/frr/ospfd.conf` anpassen. Die Befehle hierzu finden sich in der Dokumentation zur FRR-Suite.

v) Zeigen Sie mit `traceroute` den Weg einer Nachricht von Ihrem PC3 zu 10.11.2.100 und 10.12.2.100. Über welche AS und welche Router wird die Nachricht vermittelt?

vi) Angenommen, der Lehrstuhl verlangt hohe Gebühren für den Transit Ihrer Nachrichten. Konfigurieren Sie Ihren Router1 so, dass über dieses AS nur vermittelt wird, wenn keine andere Verbindung zur Verfügung steht.
   Optional: Weisen Sie nach, das Ihre Konfiguration funktioniert, indem Sie mit den andern Gruppen kommunizieren und den Transit über diese abwickeln.

vii) Konfigurieren Sie nun Router4 anstatt Router1 als BGP-Router Ihres AS, wobei die externen AS weiterhin an Router1 angeschlossen bleiben.
   Erläutern Sie anhand Ihrer Beobachtungen, inwiefern sich die beiden Szenarien unterscheiden.
   Theorie: Nennen Sie mindestens 2 Gründe, die für ein solches Szenario sprechen würden.

◇

Literatur

[RFC 1058] HEDRICK, C.L.: *Routing Information Protocol*. RFC 1058 (Historic), Juni 1988, http://www.ietf.org/rfc/rfc1058.txt . Updated by RFCs 1388, 1723.

[RFC 1519] FULLER, V., T. LI, J. YU und K. VARADHAN: *Classless Inter-Domain Routing (CIDR): an Address Assignment and Aggregation Strategy*. RFC 1519 (Proposed Standard), September 1993, http://www.ietf.org/rfc/rfc1519.txt . Obsoleted by RFC 4632.

[RFC 1881] IAB und IESG: *IPv6 Address Allocation Management*. RFC 1881 (Informational), Dezember 1995, http://www.ietf.org/rfc/rfc1881.txt .

[RFC 2401] KENT, S. und R. ATKINSON: *Security Architecture for the Internet Protocol*. RFC 2401 (Proposed Standard), November 1998, http://www.ietf.org/rfc/rfc2401.txt . Obsoleted by RFC 4301, updated by RFC 3168.

[RFC 2460] DEERING, S. und R. HINDEN: *Internet Protocol, Version 6 (IPv6) Specification*. RFC 2460 (Draft Standard), Dezember 1998, http://www.ietf.org/rfc/rfc2460.txt . Updated by RFC 5095.

[RFC 2784] FARINACCI, D., T. LI, S. HANKS, D. MEYER und P. TRAINA: *Generic Routing Encapsulation (GRE)*. RFC 2784 (Proposed Standard), März 2000, http://www.ietf.org/rfc/rfc2784.txt . Updated by RFC 2890.

[RFC 3315] DROMS, R., J. BOUND, B. VOLZ, T. LEMON, C. PERKINS und M. CARNEY: *Dynamic Host Configuration Protocol for IPv6 (DHCPv6)*. RFC 3315 (Proposed Standard), Juli 2003, http://www.ietf.org/rfc/rfc3315.txt . Updated by RFCs 4361, 5494.

[RFC 4271] REKHTER, Y., T. LI und S. HARES: *A Border Gateway Protocol 4 (BGP-4)*. RFC 4271 (Draft Standard), Januar 2006, http://www.ietf.org/rfc/rfc4271.txt .

[RFC 4361] LEMON, T. und B. SOMMERFELD: *Node-specific Client Identifiers for Dynamic Host Configuration Protocol Version Four (DHCPv4)*. RFC 4361 (Proposed Standard), Februar 2006, http://www.ietf.org/rfc/rfc4361.txt . Updated by RFC 5494.

[RFC 4443] CONTA, A., S. DEERING und M. GUPTA: *Internet Control Message Protocol (ICMPv6) for the Internet Protocol Version 6 (IPv6) Specification*. RFC 4443 (Draft Standard), März 2006, http://www.ietf.org/rfc/rfc4443.txt . Updated by RFC 4884.

[RFC 4884] BONICA, R., D. GAN, D. TAPPAN und C. PIGNATARO: *Extended ICMP to Support Multi-Part Messages*. RFC 4884 (Proposed Standard), April 2007, http://www.ietf.org/rfc/rfc4884.txt .

[RFC 5095] ABLEY, J., P. SAVOLA und G. NEVILLE-NEIL: *Deprecation of Type 0 Routing Headers in IPv6*. RFC 5095 (Proposed Standard), Dezember 2007, http://www.ietf.org/rfc/rfc5095.txt .

[RFC 5398] HUSTON, G.: *Autonomous System (AS) Number Reservation for Documentation Use*. RFC 5398 (Informational), Dezember 2008, http://www.ietf.org/rfc/rfc5398.txt .

[RFC 5494] ARKKO, J. und C. PIGNATARO: *IANA Allocation Guidelines for the Address Resolution Protocol (ARP)*. RFC 5494 (Proposed Standard), April 2009, http://www.ietf.org/rfc/rfc5494.txt .

[RFC 791] POSTEL, J.: *Internet Protocol*. RFC 791 (Standard), September 1981, http://www.ietf.org/rfc/rfc791.txt . Updated by RFC 1349.

[WWW 18] KUNIHIRO ISHIGURO, ET AL.: *FRRouting Documentation*, Dezember 2018, http://docs.frrouting.org/en/latest/ .

INSTITUT FÜR
INFORMATIK
DER LUDWIG-MAXIMILIANS-UNIVERSITÄT MÜNCHEN

LS Prof. Kranzlmüller

Praktikum Rechnernetze

Kapitel 4: Software Defined Networks

MNM
TEAM
Munich Network Management Team

4 Software Defined Networks

Inhaltsverzeichnis

4.1 Introduction 63
4.2 Architecture 65
4.2.1 Components 66
4.2.2 Interfaces 68
4.2.3 SDN Applications 68
4.3 Flows 69
4.4 SDN Deployment Example with P4 and P4Runtime 71
4.4.1 P4 71
4.4.2 P4Runtime 75
4.4.3 Test-bed 77
4.4.4 Example: deploying a simple repeater 78
4.4.5 Other P4 sources and control applications for references . . 81
4.5 Assignment 82
4.5.1 Scenario 82
4.5.2 Demo 83
4.5.3 Submission 84
4.5.4 Assessment 84
4.5.5 Important dates 84

This chapter introduces Software-defined Networks (SDN) as a new networking approach in contrast to traditional networks. In order to understand the principles of SDN, the layer views of network functionality in traditional networks and in SDN are juxtaposed, the SDN architecture is then presented. The concepts of *flow* commonly used in the SDN literature is analysed in this text. A deployment example with P4-based SDN is followed. The exercise illustrates how to employ SDN to "program" a network, revealing a new way of performing network control and management.

4.1 Introduction

There are different ways to layer networked systems. Two well-known ones are the ISO OSI (Open Systems Interconnection) and the TCP/IP reference models. Figure 4.1 shows an example of a switch and a router through the OSI view. The switch spans two OSI layers, namely Physical and Data Link whereas the router implements an additional Network layer. A network device is classified into its highest layer, which means that

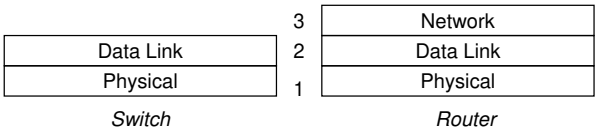


Figure 4.1: Switch and Router in OSI model [Danc 15]

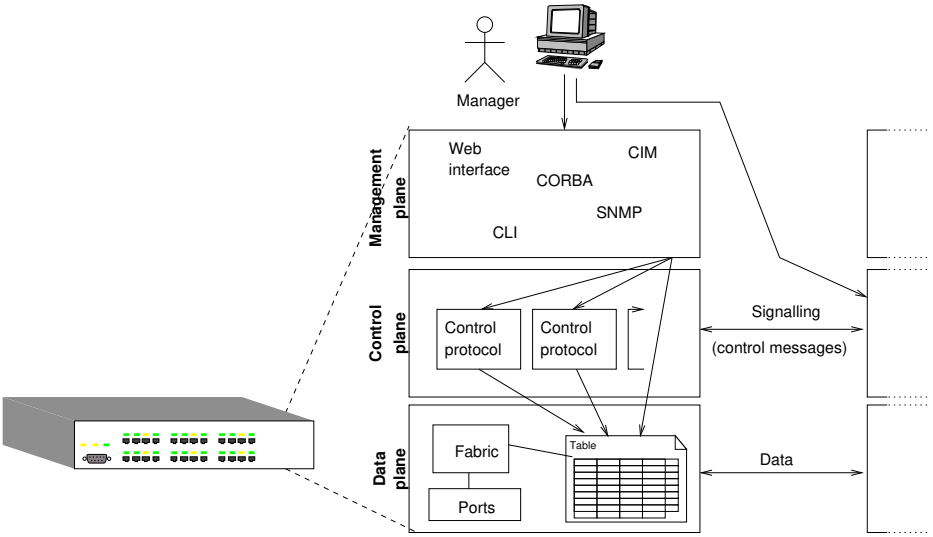


Figure 4.2: Layer view of networking device's functionality [Danc 15]

the switch belongs to the Data Link layer and the router to the Network layer. The OSI layering mechanism provides a common basis for the coordination of networking standards' development as well as facilitates the understanding of these standards and the interaction of networked systems.

In another view, a traditional network device can be seen as being composed of the three planes (or layers) depicted in Figure 4.2: management plane, control plane and data plane.

- The data plane consists of various ports for receiving and transmitting packets based on its forwarding table, and switching fabrics for transferring packets from an input buffer to an output buffer.
- The control plane represents protocols used for populating forwarding tables in the data plane, e.g., the routing protocols like RIP, OSPF, BGP in a router.
- The management plane includes software services used by a network administrator (manager) to monitor and configure the control functionalities.

Figure 4.3 describes the roles of these planes and their interactions.

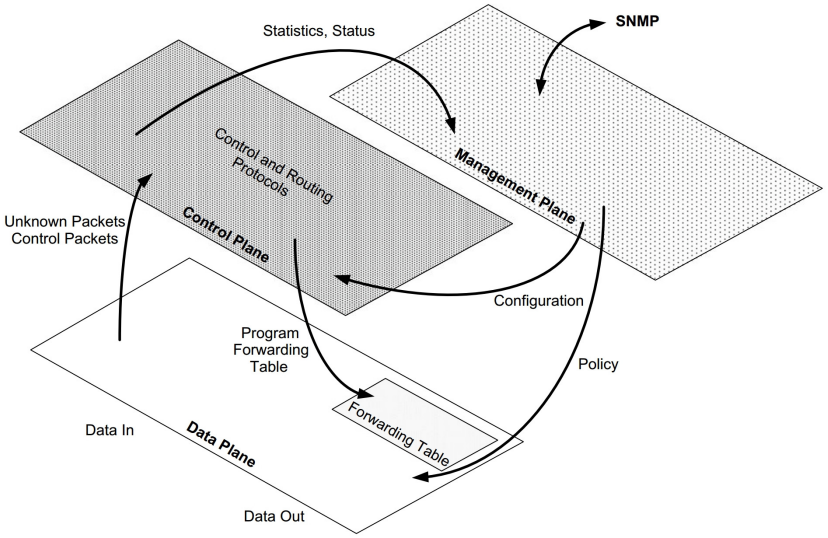


Figure 4.3: Roles of the management, control and data planes [GoBl 14]

Although traditional IP networks have widespread adoption, they are complex and hard to manage. To make any change to a network, operators need to configure each individual network device separately using low-level and vendor-specific commands. The vertical integration of the control and data plane in each network device reduces the flexibility and hinders the innovation and evolution of networking infrastructure. These limitations pose a question of new networking paradigms, leading to the introduction of Software-Defined Networking (SDN).

Some key ideas of SDN are the introduction of dynamic programmability in forwarding devices, the decoupling of the control and data planes, and the global view of the network by logical centralization of the "network brain" [KRV+ 15] in a single place, namely the controller.

4.2 Architecture

Having explained the three-plane view of traditional networks, we will now go deeper into the SDN architecture.

SDN is a network architecture where network control is decoupled from forwarding and is directly programmable [Foun 12].

A comparison of a traditional network and a SDN in the three plane view is illustrated in Figure 4.4. In the traditional network, the routing table at the data plane of a router is populated by its control plane residing in the same device, which is the OSPF routing protocol running on the router's Operating System in this example. In SDN, the controller

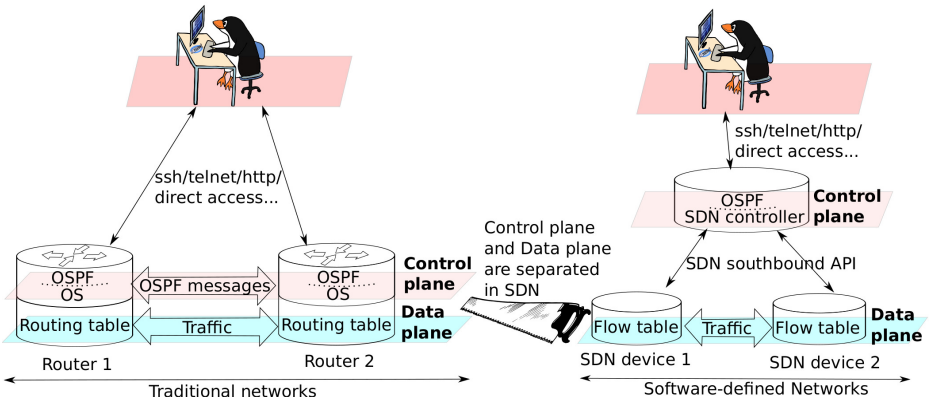


Figure 4.4: An example of functionality plane mapping in traditional networks and in SDN

plays the role of a "network operating system" and the OSPF routing protocol runs atop the controller, which asks the controller to populate flow tables in the SDN devices via the southbound API. The mapping of the management plane's services in these two networks is not relevant in terms of network control and is not portrayed in this example to avoid unnecessary confusion.

The general SDN architecture is illustrated in Figure 4.5.

Kreutz et al. [KRV+ 15] define SDN as a network architecture with four pillars.

1. The control and data planes are decoupled. Control functionality is removed from network devices that will become simple (packet) forwarding elements.

2. Forwarding decisions are flow-based, instead of destination-based (refer to Section 4.3 for the explanation of the *flow* concept).

3. Control logic is moved to an external entity, the so-called SDN controller.

4. The network is programmable through software applications running on top of the controller that interacts with underlying data plane devices.

Another important characteristic of SDN mentioned in [RFC 7426] is the standardization of the interfaces between the control and data planes.

4.2.1 Components

The main components of a SDN include devices in the data plane and one or more controllers.

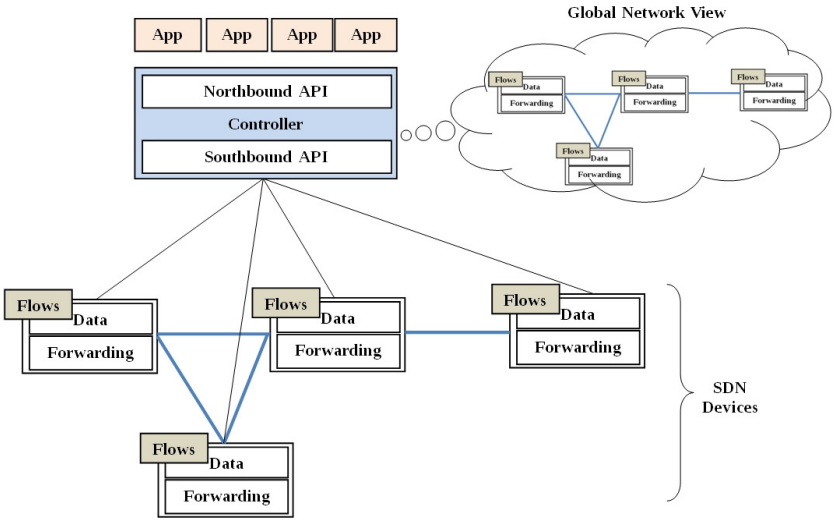


Figure 4.5: General architecture of SDN [GoBl 14]

SDN-devices

SDN devices, also commonly referred to as SDN switches, are simple forwarding elements without embedded control or software to take autonomous decisions like routing protocols, default/fixed forwarding behaviour. The network intelligence is removed from them to a logically centralized controller. Packets are handled in these switches based on their flow tables, whose entries contain control information of different layers (e.g., layer 2 - 4). Standardized interfaces are introduced between the controller and the switches, which provide a means for the controller to install flow tables in these switches.

Controller

Controller is a software stack that controls SDN-devices. One important function of the controller is the provision of topology service, which maintains the consistent overview of the network. Any change in the network, such as new devices are added or some devices are removed, some links are down, should be reflected immediately in the network overview at the controller. The controller changes configuration of network devices based on applications' requests.

According to Goransson et al. [GoBl 14], four fundamental functions of a SDN-controller are:

- end-user device discovery,
- network device discovery,
- network device topology management, which maintains information about the

interconnection details of the network devices to each other and to the end-user devices to which they are directly attached,

- flow management, which maintains a database of the flows being managed by the controller and performs all necessary coordination with the devices to ensure synchronization of the device flow entries with that database.

Further functions of the network are realized by SDN-applications (see Section 4.2.3).

4.2.2 Interfaces

The SDN architecture introduces two interfaces: the northbound APIs lying between the applications and the controller, and the southbound APIs between the controller and the SDN-devices. While OpenFlow [MAB+ 08] and P4Runtime [P4RTSpec] appear to be the most dominant soundbound APIs, there is no such counterparts for northbound APIs so far. Each controller has its own northbound APIs.

4.2.3 SDN Applications

The control functions of the network (MAC learning of switches, routing, enforcement of QoS, security...) are programmed in applications, which logically reside above the controller. Applications can change configuration of switches via the controller's northbound API. Taking a simple routing application as an example [KRV+ 15]: the logic of this application is to define the path through which packets flow from a point A to a point B. To achieve this goal, the routing application has to, based on the topology input, decide on the path to use and instruct the controller to install the respective forwarding rules in all devices on the chosen path, from A to B.

Once the controller has finished initializing devices and has reported the network topology to the application, the application spends most of its processing time responding to events. Application behavior is driven by events coming from the controller as well as external inputs. The application affects the network by responding to the events as modeled in Figure 4.6. The SDN application registers as a listener for certain events, and the controller invoke the application's callback method whenever such an event occurs. Some examples of events handled by a SDN application are end-user device discovery, network device discovery, and incoming packets. In the first two cases, events are sent to the SDN application upon the discovery of a new end-user device (i.e., a MAC address) or a new network device (e.g., a switch, router, or wireless access point), respectively. Incoming packet events are sent to the SDN application when a packet is received from a SDN device due to either a flow entry instructing the SDN device to forward the packet to the controller, or because there is no matching flow entry in the SDN device. When there is no matching flow entry, the default action is usually to forward the packet to the controller, though it could be to drop the packet, depending on the nature of the applications [GoBl 14].

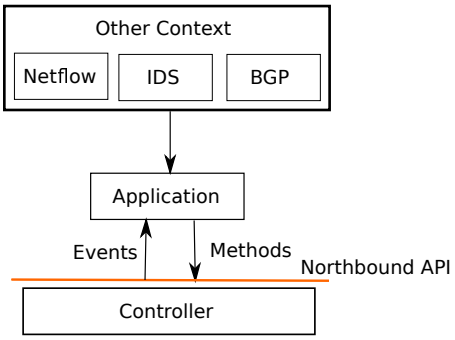


Figure 4.6: Application-Controller communication [GoBl 14]

4.3 Flows

In SDN, forwarding decisions are flow-based, instead of destination-based as in traditional networks. There is usually confusion between the two different concepts: *path* and *flow*, they need to be distinguished and used correctly.

A path is the sequence of communication links and devices connected two endpoints. It is also known as a route and is the fundamental of IP-routing. The concept of *path* is topology-oriented, which means it is only concerned with the way through the network, not with the traffic type and content (payload).

A flow or a data stream, is a sequence of packets which share the same attributes. The mentioned attributes include the fields in the protocol header, such as IP address, MAC address, status bit...; they can also be the ingress port of a packet coming to a network device. Therefore, the concept of *flow* includes traffic types and possibly even traffic content (payload). Flows are unidirectional. A flow is thus determined by:

- the path,
- the concerned attributes,
- the direction (can be considered as a special case of the attribute: *address*).

Example: traffic between two endpoints A and B that are exchanging audio stream and HTTP traffic can be seen as four different flows:

1. a flow for audio traffic along the path from A to B,
2. a flow for audio traffic along the path from B to A,
3. a flow for HTTP traffic along the path from A to B,
4. a flow for HTTP traffic along the path from B to A.

Figure 4.7 [Danc 15] illustrates how four data streams can be routed in a traditional network (the left picture) and in a SDN (the right picture). In the traditional network,

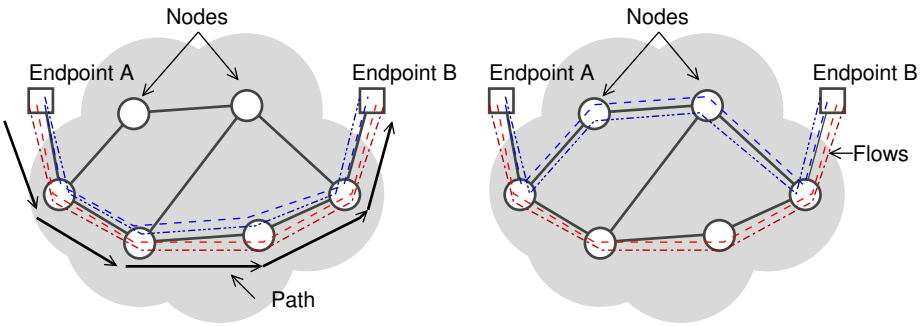


Figure 4.7: Flows along different paths in the traditional network (left) and in SDN [Danc 15]

the data streams destined to the same destination are normally routed along the same path since the routing mechanism is destination-based. In SDN, these data streams are routed based on their flows' attributes. As a consequence, they may traverse different paths from the source to the destination.

Note that, a flow can be reflected differently in each device along that path. For example, one network device handles the flow based on the first attribute (e.g., destination IP address) of that flow while the other based on the second attribute (e.g., destination TCP port number).

Specification of Flow A SDN-device keeps the information of flows in tables, named flow tables or rule tables. Each table entry includes match fields, which specify the flow based on protocol fields (IP, MAC, port, VLAN...) and actions, being executed when the match fields are satisfied.
A match field may be wildcarded to match any value. By this way, the number of table entries can be reduced.
The action applying to a matched flow can be:

- sending the packets of that flow to a specific port (interface of the SDN-device),
- dropping its packets,
- forwarding its packets to the controller,
- sending them to some or all ports (flooding, broadcast, multicast),
- modifying its packet,
- a certain combination of the above actions.

Priority Table entries are organized by rows, each with its specified priority. Therefore, specific entries should have higher priority than more general one (which have more masked fields).

4.4 SDN Deployment Example with P4 and P4Runtime

The most common implementations of SDN include those based on OpenFlow [MAB+ 08] and P4 [BDG+ 14]. OpenFlow assumes that SDN devices have fixed behaviour, it supports populating the rule tables in these devices constrained by their fixed capabilities. It is impossible to use OpenFlow to deploy a rule that can influence a new packet header not yet available in these fixed-function devices. P4 was introduced to address this problem by providing a means for programming devices' behaviour. P4Runtime enables the communication between SDN controllers and P4-devices.

4.4.1 P4

P4 is a language for programming the data plane of network devices. The name P4 comes from the original paper that introduced the language: "Programming Protocol-independent Packet Processors" [BDG+ 14]. The version of P4 introduced in 2014 is named $P4_{14}$, the language has been revised and the revision in 2016 is named $P4_{16}$, which is used in this assignment.

In a traditional switch the manufacturer defines the data-plane functionality. The control plane controls the data plane by managing entries in their rule tables (e.g., routing tables), configuring specialized objects (e.g., meters), and by processing control packets (e.g., routing protocol packets) or asynchronous events, such as link state changes or learning notifications.

A P4-programmable switch differs from a traditional switch in two essential ways:

- The data plane functionality is not fixed in advance but is defined by a P4 program. The data plane is configured at initialization time to implement the functionality described by the P4 program and has no built-in knowledge of existing network protocols.
- The control plane communicates with the data plane using the same channels as in a fixed-function device, but the set of tables and other objects in the data plane are no longer fixed, since they are defined by a P4 program. The P4 compiler generates the API that the control plane uses to communicate with the data plane.

Figure 4.8 shows a typical tool workflow when programming a target using P4. A target is defined as a packet-processing system capable of executing a P4 program. The term *target* is used interchangeably with the term *device* in most P4 specifications and also in this manuscript.

Target manufacturers provide the hardware or software implementation framework, an architecture definition, and a P4 compiler for that target. P4 programmers write programs for a specific architecture, which defines a set of P4-programmable components on the target as well as their external data plane interfaces.

Compiling a set of P4 programs produces two artifacts:

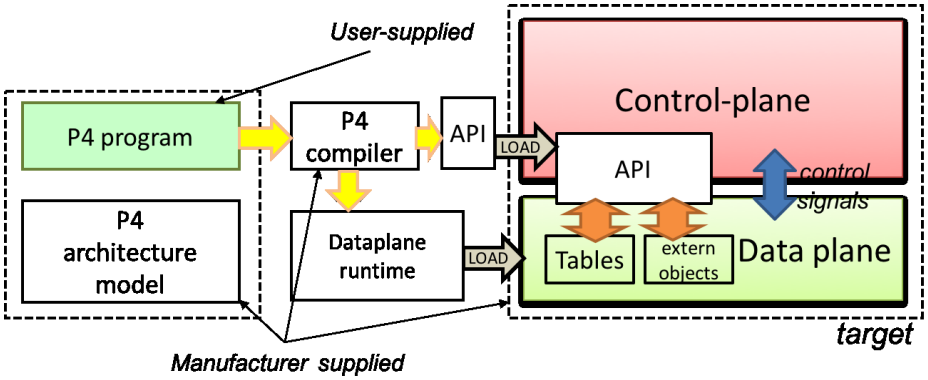


Figure 4.8: Programming a target with P4 [P4Spec]

- a data plane configuration that implements the forwarding logic described in the input program and
- an API for managing the state of the data plane objects from the control plane.

Architecture Model

The P4 architecture identifies the P4-programmable blocks (e.g., parser, ingress control flow, egress control flow, deparser, etc.) and their data plane interfaces.

The P4 architecture can be thought of as a contract between the program and the target. Each manufacturer must therefore provide both a P4 compiler as well as an accompanying architecture definition for their target.

Figure 4.9 illustrates the data plane interfaces between P4-programmable blocks. It shows a target that has two programmable blocks (#1 and #2). Each block is programmed through a separate fragment of P4 code. The target interfaces with the P4 program through a set of control registers or signals. Input controls provide information to P4 programs (e.g., the input port that a packet was received from), while output controls can be written to by P4 programs to influence the target behavior (e.g., the output port where a packet has to be directed). Control registers/signals are represented in P4 as intrinsic metadata. P4 programs can also store and manipulate data pertaining to each packet as user-defined metadata.

There exist various architectures, e.g., V1Model[1], SimpleSumeSwitch[2], Portable Switch Architecture (PSA) [PSADraft]. In this assignment, we employ V1Model on the BMv2 (Behavioral Model version 2) Simple Switch target[3] due to their ease of deployment as a software switch.

[1] https://github.com/p4lang/p4c/blob/main/p4include/v1model.p4
[2] https://github.com/NetFPGA/P4-NetFPGA-public/wiki
[3] https://github.com/p4lang/behavioral-model/blob/main/docs/simple_switch.md

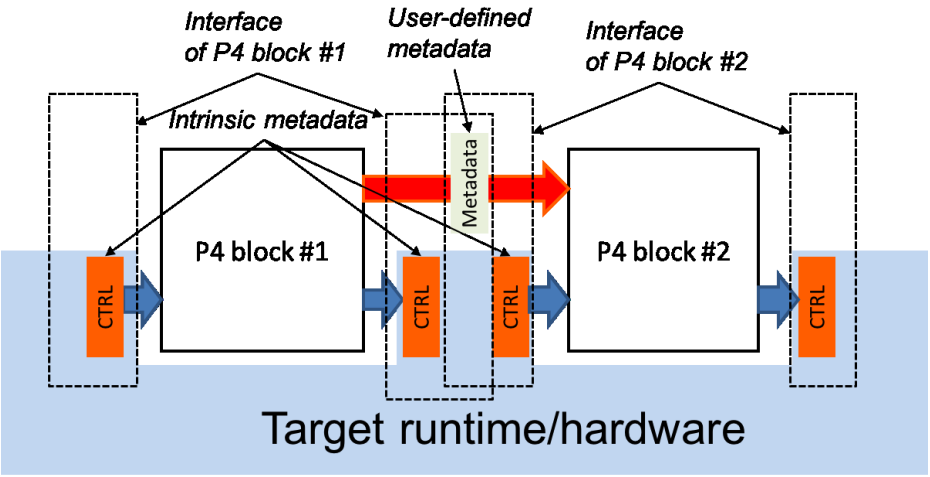


Figure 4.9: P4 program interfaces

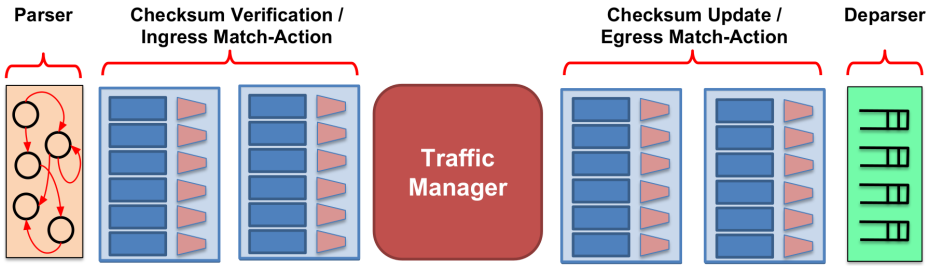


Figure 4.10: V1Model architecture

V1Model

Figure 4.10 depicts the programmable blocks of the V1Model architecture. Its standard metadata includes:

```
struct standard_metadata_t {
    bit<9> ingress_port;
    bit<9> egress_spec;
    bit<9> egress_port;
    bit<32> clone_spec;
    bit<32> instance_type;
    bit<1> drop;
    bit<16> recirculate_port;
    bit<32> packet_length;
    bit<32> enq_timestamp;
    bit<19> enq_qdepth;
    bit<32> deq_timedelta;
```

```
    bit<19> deq_qdepth;
    bit<48> ingress_global_timestamp;
    bit<32> lf_field_list;
    bit<16> mcast_grp;
    bit<1> resubmit_flag;
    bit<16> egress_rid;
    bit<1> checksum_error;
    bit<3> priority;
}
```

The *ingress_port* is the port on which a packet arrived, the *egress_spec* specifies the port to which that packet should be sent to, the *egress_port* denotes the port on which the packet is departing from (read-only in egress pipeline). The file v1model.p4[4] is extensively commented, which provides details also on other fields of the standard metadata.

The $P4_{16}$ program template for the V1Model architecture include the *HEADER* specification, the *PARSER, CHECKSUM VERIFICATION, INGRESS PROCESSING, EGRESS PROCESSING, CHECKSUM UPDATE, DEPARSER* blocks, and the *V1Switch* package.

```
#include <core.p4>
#include <v1model.p4>
/* HEADERS */
struct metadata { ... }
struct headers {
    ethernet_t ethernet;
    ipv4_t ipv4;
}
/* PARSER */
parser MyParser(packet_in packet,
        out headers hdr,
        inout metadata meta,
        inout standard_metadata_t smeta) {
    ...
}
/* CHECKSUM VERIFICATION */
control MyVerifyChecksum(in headers hdr,
                         inout metadata meta) {
    ...
}
/* INGRESS PROCESSING */
control MyIngress(inout headers hdr,
                  inout metadata meta,
                  inout standard_metadata_t std_meta) {
    ...
}
/* EGRESS PROCESSING */
```

[4] https://github.com/p4lang/p4c/blob/main/p4include/v1model.p4

```
control MyEgress(inout headers hdr,
                 inout metadata meta,
                 inout standard_metadata_t std_meta) {
    ...
}
/* CHECKSUM UPDATE */
control MyComputeChecksum(inout headers hdr,
                          inout metadata meta) {
    ...
}
/* DEPARSER */
control MyDeparser(inout headers hdr,
                   inout metadata meta) {
    ...
}
/* SWITCH */
V1Switch(
    MyParser(),
    MyVerifyChecksum(),
    MyIngress(),
    MyEgress(),
    MyComputeChecksum(),
    MyDeparser()
) main;
```

$P4_{16}$ defines various data types, e.g., *bit, bool, int, string, struct, enum, header_union*. The $P4_{16}$ data types are described in its specification [P4Spec] (c.f. Section 7 of this specification). Another useful reference source is the *P4 Language Cheat Sheet*[5], which outlines the basic building blocks of P4.

An example of compiling a P4 source into a *P4 Device Config* file to be executed in BMv2 switches is shown in a later section.

4.4.2 P4Runtime

The P4Runtime API [P4RTSpec] is a control plane specification for controlling the data plane elements of a device defined or described by a P4 program. Figure 4.11 represents the P4Runtime reference architecture. The device or target to be controlled is at the bottom, and one or more controllers is shown at the top. P4Runtime only grants write access to a single primary controller for each read/write entity.

The P4Runtime API defines the messages and semantics of the interface between the client(s) and the server. The API is specified by the p4runtime.proto Protobuf file[6].

The controller can access the P4 entities which are declared in the P4Info metadata. The

[5] https://github.com/p4lang/tutorials/blob/master/p4-cheat-sheet.pdf
[6] https://github.com/p4lang/p4runtime

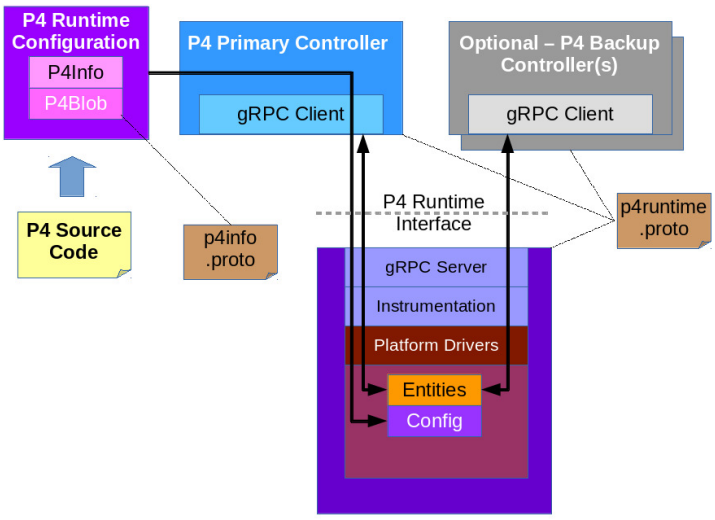


Figure 4.11: P4Runtime reference architecture [P4RTSpec]

P4Info structure is defined by p4info.proto, another Protobuf file available as part of the standard.

The controller can also set the *ForwardingPipelineConfig*, which amounts to installing and running the compiled P4 program output included in the *p4_device_config* Protobuf message field, and installing the associated P4Info metadata. Furthermore, the controller can query the target for the *ForwardingPipelineConfig* to retrieve the device config and the P4Info.

The P4Runtime API is implemented by a program that runs a gRPC server which binds an implementation of auto-generated P4Runtime Service interface. This program is called the "P4Runtime server". The server must listen on TCP port 9559 by default, which is the port that has been allocated by IANA for the P4Runtime service. Servers should allow users to override the default port using a configuration file or flag when starting the server.

In this assignment, we follow the "idealized workflow" mentioned in the P4Runtime specification [P4RTSpec] (see Section 3.2 of this specification), in which a P4 source program is compiled to produce both a P4 device config and P4Info metadata. These comprise the *ForwardingPipelineConfig message*. A P4Runtime controller chooses a configuration appropriate to a particular target and installs it via a *SetForwardingPipelineConfig* RPC. Metadata in the P4Info describes both the overall program itself (PkgInfo) as well as all entity instances derived from the P4 program — tables and extern instances. Each entity instance has an associated numeric ID assigned by the P4 compiler which serves as a concise "handle" used in API calls.

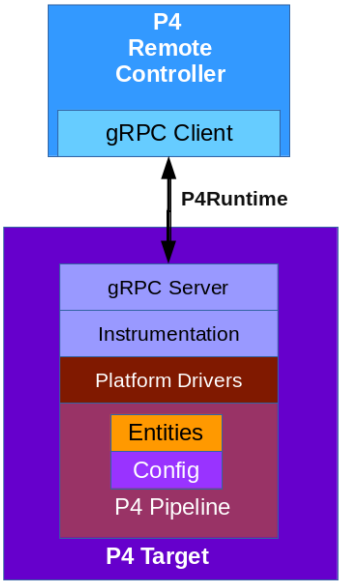


Figure 4.12: Single remote controller [P4RTSpec]

In this workflow, P4 compiler backends are developed for each unique type of target and produce P4Info and a target-specific device config. The P4Info schema is designed to be target- and architecture-independent, although the specific contents are likely to be architecture-dependent. The compiler ensures that the code is compatible with the specific target and rejects code which is incompatible.

P4Runtime allows for more than one controller. A controller can be embedded in a P4 target or separate therefrom. For P4-based SDN, we opt for the deployment of separate controllers from targets as shown in Figure 4.12, and a single logically centralized controller can control multiple targets at the same time.

4.4.3 Test-bed

The provided infrastructure for this assignment includes one “outer” virtual machine (gruppeX.rnp.lab.nm.ifi.lmu.de) on which seven “inner” virtual machines (including hosts pc1, pc2, pc3, switches S1, S2, S3 and a controller deployed at router4) are created. Their connections are sketched in Figure 4.13; the dashed lines indicate the management network used for the access and configuration of the inner machines, the solid lines represent the data network in which communication between the inner machines is carried out. All *eth0* interfaces of the inner machines are connected to the same switch of the management network.

The infrastructure has been prepared with all software necessary for this assignment. The

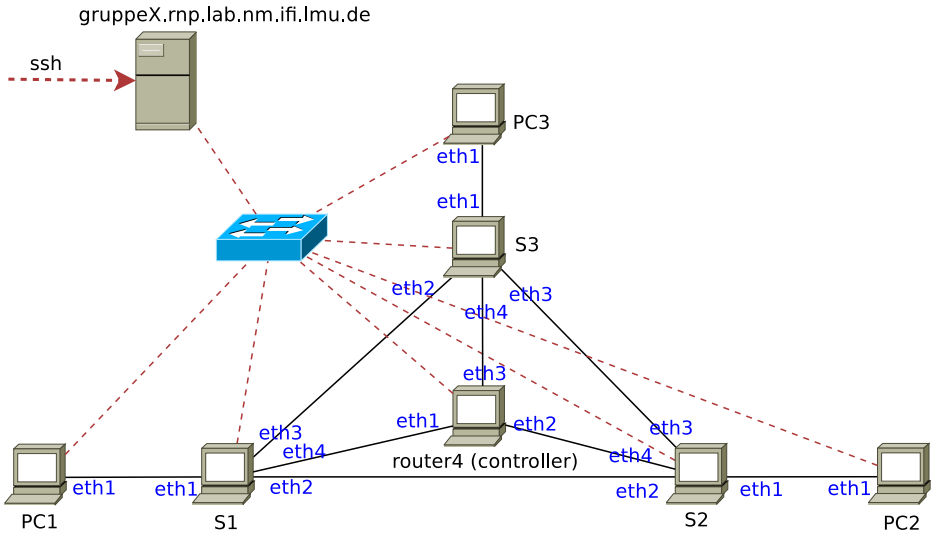


Figure 4.13: Provided RNP infrastructure

P4 compiler was also installed at the controller, i.e., at router4 (and only there). This means, a P4 source file needs to be compiled at the controller to obtain the *P4 Device config* and the *P4Info* files, which are to be consumed by P4 switches and controllers, respectively.

Instructions for installing P4-related software are available in the P4 language tutorials[7], in which a sample deployment on Ubuntu 20.04 is provided[8]. The development team also facilitates the P4 installation via a package manager for Debian 11, Ubuntu 22.04 thanks to the *p4lang* repository[9]. The installation can also be carried out directly from source code[10].

4.4.4 Example: deploying a simple repeater

We demonstrate how to execute P4 switches and have them controlled by a controller based on a simple repeater application. This example reuses the existing material from the ETH Networked Systems Group[11].

Figure 4.14 shows the network topology used in this example, including two switches S1, S2, two end-points PC1, PC2 and a controller.

[7] https://github.com/p4lang/tutorials
[8] https://github.com/p4lang/tutorials/blob/master/vm-ubuntu-20.04/root-release-bootstrap.sh
[9] http://download.opensuse.org/repositories/home:/p4lang/
[10] https://github.com/p4lang/tutorials/blob/master/vm-ubuntu-20.04/user-dev-bootstrap.sh
[11] https://github.com/nsg-ethz/p4-learning/tree/master/exercises/02-Repeater/p4runtime

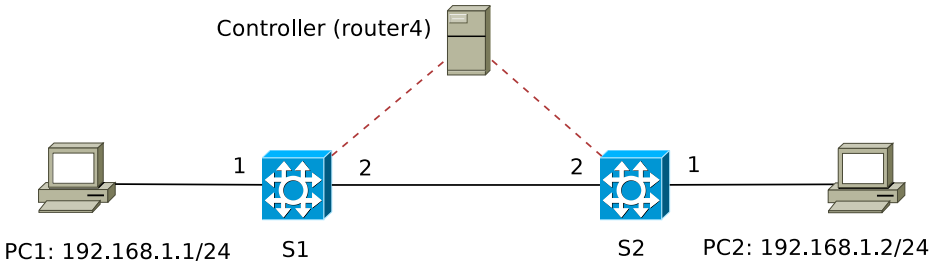


Figure 4.14: Network topology for the deployment example. The numbers surrounding a switch indicate its port names.

The P4 source files and the controller applications are placed at the controller (router4) in the directories *p4* and *app*, respectively.

Compiling repeater.p4

As previously mentioned, the p4 compiler is installed at the controller, which is *router4* in our case. We need to compile *repeater.p4* to obtain the *P4 Device config* and the *P4Info* files as follows:

```
ssh root@router4
cd p4
p4c-bm2-ss --p4v 16 --p4runtime-files build/repeater.p4info.txt
           -o build/repeater.json repeater.p4
```

The *P4 Device config* file *repeater.json* and the *P4Info* file *repeater.p4info.txt* are consumed by the controller application *repeater_controller.py*. The *P4 Device config* file *repeater.json* is used for the P4 switches, we need to copy this file to the switches S1 and S2. This can be done simply from the outer machine with these commands:

```
scp -3 root@router4:p4/build/repeater.json root@s1:
scp -3 root@router4:p4/build/repeater.json root@s2:
```

Creating P4 switches

We can create a P4 switch at switch S1 by the following commands:

```
ssh root@s1
ip link set eth1 up
ip link set eth2 up
simple_switch_grpc  -i 1@eth1 -i 2@eth2 --pcap pcaps
      --nanolog ipc:///log.ipc --device-id 1 repeater.json
      --log-console --thrift-port 9090
      -- --grpc-server-addr 0.0.0.0:50051 --cpu-port 255
```

The command *simple_switch_grpc* creates a SimpleSwitchGrpc target[12], which is a version of SimpleSwitch with P4Runtime support.

- the part *-i 1@eth1 -i 2@eth2* specifies the binding of the switch ports to the "physical" interfaces of the (virtual) machine,
- *--pcap pcaps*: generating pcap files for interfaces,
- *--nanolog ipc:///log.ipc*: IPC socket to use for nanomsg pub/sub logs,
- *--device-id 1*: device ID, used to identify the device in IPC messages,
- *--log-console*: enabling logging on *stdout*,
- *--thrift-port 9090*: TCP port on which to run the Thrift runtime server,
- *--grpc-server-addr 0.0.0.0:50051*: bind gRPC server to the given address,
- *--cpu-port 255*: the logical port where a switch can send a packet to a controller (packet-in), or a controller can send a packet to a switch (packet-out).

More information can be found in the command *simple_switch_grpc --help*.

Similarly, we can create a P4 switch at switch S2:

```
ssh root@s2
ip link set eth1 up
ip link set eth2 up
simple_switch_grpc -i 1@eth1 -i 2@eth2 --pcap pcaps
      --nanolog ipc:///log.ipc --device-id 2 repeater.json
      --log-console --thrift-port 9090
      -- --grpc-server-addr 0.0.0.0:50051 --cpu-port 255
```

Executing the control application

Now we can execute the control application:

```
ssh root@router4
cd app
python3 repeater_controller.py
```

We use the *p4-utils* library[13] with some modifications for *packet-in, packet-out, idle timeout* support to implement control applications. *p4-utils* can be considered as a wrapper of *p4runtime-shell*[14] for easier development of the controller plane. Useful information of the APIs, e.g., *table_add, table_delete_match...* can be found in the mentioned *GitHub* link[15].

[12] https://github.com/p4lang/behavioral-model/blob/main/targets/simple_switch_grpc/README.md
[13] https://github.com/nsg-ethz/p4-utils
[14] https://github.com/p4lang/p4runtime-shell
[15] https://nsg-ethz.github.io/p4-utils/p4utils.utils.sswitch_p4runtime_API.html

Note that the *repeater* application uses the network topology information encoded in *topo_repeater.json*. For other control applications, we need to provide relevant information of the network topology in a similar file.

Generating traffic between end-points

Once the *repeater* application has been deployed, we can generate the traffic between PC1 and PC2 to test its functionality.

Configuring the *eth1* interface of PC1:

```
ssh root@pc1
ip addr add 192.168.1.1/24 broadcast 192.168.1.255 dev eth1
ip link set eth1 up
```

Configuring the *eth1* interface of PC2 and generating traffic via the *ping* command:

```
ssh root@pc2
ip addr add 192.168.1.2/24 broadcast 192.168.1.255 dev eth1
ip link set eth1 up
ping 192.168.1.1
```

If the PCs can see each other via *ping*, then the deployment of the *repeater* application was successful.

Using *simple_switch_CLI* for data plane debugging

The command *simple_switch_CLI* is useful for debugging purpose in the data plane. For example, while switch S1 is running, we can open another terminal window to observe or manipulate its rule tables as follows:

```
ssh root@s1
simple_switch_CLI
show_tables   #show the existing tables in this switch
table_dump repeater #show the contents of the rule table named repeater
table_add repeater forward 1 => 2 # add to table repeater a rule,
                # match key = 1, action = forward, action parameter = 1
```

More commands and their details are described at the p4language GitHub link[16].

4.4.5 Other P4 sources and control applications for references

An advanced control application, named *arpcache.py*, is provided in *router4* together with the above example, its corresponding P4 source is *packetinout.p4*.

[16] https://github.com/p4lang/behavioral-model/blob/main/docs/runtime_CLI.md

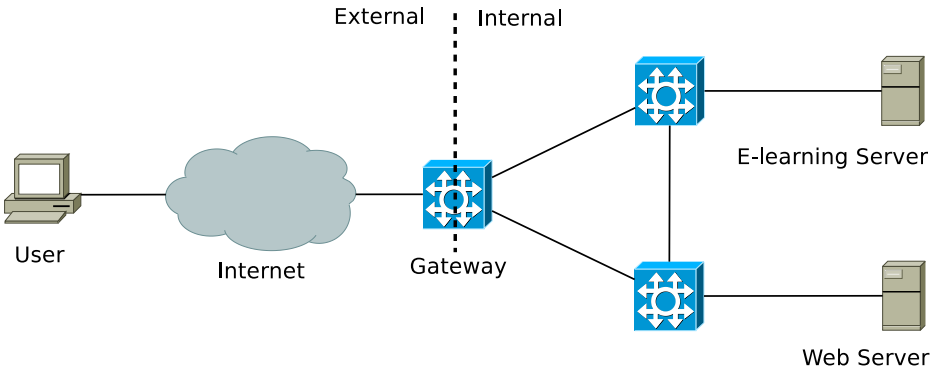


Figure 4.15: Simplified university network with two servers

We provide further useful examples in the MNM-github repository [17]. These examples should be practised in the order mentioned in the main page (file *README.md*) for better understanding and for a smooth transition between them, starting with the *simple_demo* application, then *simple_switch* and so on.

Existing P4 sources and control applications are also available at the P4 language tutorials[18] and p4-learning[19] links.

4.5 Assignment

The students will "program" the provided infrastructure using P4-based SDN to fulfill certain requirements motivated by a practical scenario.

4.5.1 Scenario

The Academic Affairs Office (AAO) of the university A has a web server running a website, so-called AAO-Website, to publish necessary information for students, and an E-learning Server providing experimental e-learning services (publishing lectures, exercises, evaluation of students' results...). Apart from the network of other faculties and other computers, servers, the network containing these two servers can be sketched roughly in Figure 4.15.

Since E-learning is still in the test phase, its load is not high and its server has spare resources most of the time. In contrast, the web server has surprisingly stressful load by the time the AAO publishes the results of the university entrance exam. The network administrator therefore decides on a temporary solution for the time being as follows:

[17] https://github.com/mnm-team/p4-sdn.git
[18] https://github.com/p4lang/tutorials/tree/master/exercises
[19] https://github.com/nsg-ethz/p4-learning/tree/master/exercises

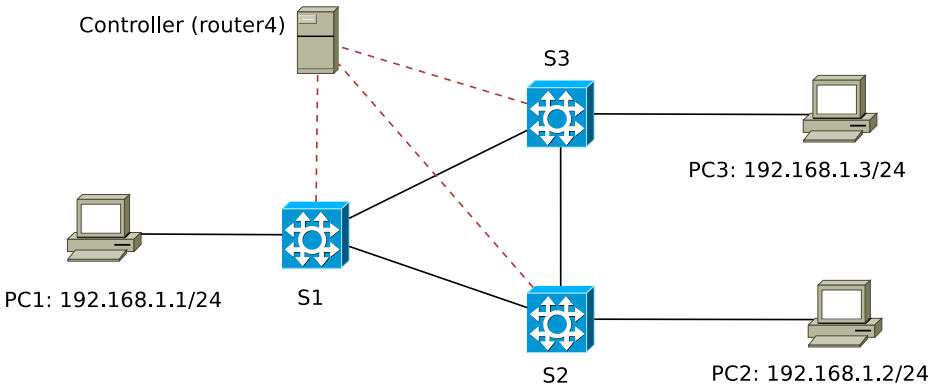


Figure 4.16: Network for deployment

- Deploying an additional instance of the AAO-Website on the E-learning Server, then performing the load balancing at the gateway to forward traffic flows alternately to these two servers.
- If the traffic accounts to 80% of the link bandwidth to these servers, drop the highest-load traffic flow.

The internal network of the university A is SDN-based.

In the role of the network administrator, the students will "program" the network to fulfill the above solution.

4.5.2 Demo

For the sake of the demo purpose, the above scenario is simplified as in Figure 4.16, PC1 plays the role of the outside party, switch S1 is the gateway, switches S2, S3 and PC2, PC3 belong to the internal network. The requirements are as follows.

- If the traffic volume from PC1 to PC2 is $\geq$ 512 Kbps, perform the load balancing at the gateway: traffic flows will alternately be delivered to PC3 and PC2 since PC3 also provides the same service as PC2. Then the response traffic from PC3 to PC1 will also be modified at the gateway: its source IP address and source MAC address are changed to those of PC2.
- (Optional) If the traffic volume from PC1 to either PC2 or PC3 amounts to $\geq$ 768 Kbps, drop the traffic flow causing the highest load.

Note: the traffic mentioned above is the one on the direction from PC1 to PC2 only, this means that the traffic on the direction from PC2 to PC1 and the other traffic are not important and can be omitted.

Student groups will implement the appropriate SDN applications, the P4 source and deploy them on the provided infrastructure to fulfill these requirements.

4.5.3 Submission

Source code and manual of how to deploy and run the applications, compressed into a zip file.

4.5.4 Assessment

The assessment of the assignment is based on the final demo at class (in the “Baracke”) and the manual.

4.5.5 Important dates

- 14.01.2025 23:59 – Submission of the source code that measures the traffic volume from PC1 to PC2 (consult the example on Counters in https://github.com/mnm-team/p4-sdn/tree/main/counters)
- 21.01.2025 13:59 – Submission of the final source code and manual
- 21.01.2025 14:00 – Demo

Literatur

[BDG+ 14] BOSSHART, PAT, DAN DALY, GLEN GIBB, MARTIN IZZARD, NICK MCKEOWN, JENNIFER REXFORD, COLE SCHLESINGER, DAN TALAYCO, AMIN VAHDAT, GEORGE VARGHESE und DAVID WALKER: *P4: Programming Protocol-Independent Packet Processors*. SIGCOMM Comput. Commun. Rev., 44(3):87–95, jul 2014, https://doi.org/10.1145/2656877.2656890 .

[Danc 15] DANCIU, VITALIAN: *Skriptum zu den einführenden Sitzungen des Masterseminars zu Trends in Netzen: software-defined networks and network function virtualization*, 2015.

[Foun 12] FOUNDATION, OPEN NETWORKING: *White paper: Software-Defined Networking:The New Norm for Networks*. White paper, April 2012.

[GoBl 14] GORANSSON, PAUL und CHUCK BLACK: *Software defined networks: a comprehensive approach*. Morgan Kaufmann, 2014.

[KRV+ 15] KREUTZ, DIEGO, FERNANDO MV RAMOS, PAULO ESTEVES VERISSIMO, CHRISTIAN ESTEVE ROTHENBERG, SIAMAK AZODOLMOLKY und STEVE UHLIG: *Software-defined networking: A comprehensive survey*. Proceedings of the IEEE, 103(1):14–76, 2015.

[MAB+ 08] MCKEOWN, NICK, TOM ANDERSON, HARI BALAKRISHNAN, GURU PARULKAR, LARRY PETERSON, JENNIFER REXFORD, SCOTT SHENKER und JONATHAN TURNER: *OpenFlow: Enabling Innovation in Campus Networks*. SIGCOMM Comput. Commun. Rev., 38(2):69–74, März 2008, https://doi.org/10.1145/1355734.1355746 .

[P4RTSpec] THE P4.ORG API WORKING GROUP: *P4Runtime Specification*, Juli 2020, https://opennetworking.org/wp-content/uploads/2020/10/P4Runtime-Spec.pdf . Version 1.2.0.

[P4Spec] THE P4 LANGUAGE CONSORTIUM: *P4$_{16}$ Language Specification*, Juli 2022, https://p4.org/p4-spec/docs/P4-16-v1.2.3.pdf . Version 1.2.3.

[PSADraft] THE P4.ORG ARCHITECTURE WORKING GROUP: *P4$_{16}$ Portable Switch Architecture (PSA)*, https://p4.org/p4-spec/docs/PSA.html . Working draft.

[RFC 7426] HALEPLIDIS, E., K. PENTIKOUSIS, S. DENAZIS, J. HADI SALIM, D. MEYER und O. KOUFOPAVLOU: *Software-Defined Networking (SDN): Layers and Architecture Terminology*. RFC 7426, IETF, Januar 2015, http://tools.ietf.org/rfc/rfc7426.txt .