

Anja Haake, Jörg M. Haake, Christian Icking, Lihong Ma

Betriebssysteme und Rechnernetze

Kurseinheit 4:
Vermittlung und Übertragung

Fakultät für
**Mathematik und
Informatik**

Inhalt

Teil I: Betriebssysteme

1	Geräte und Prozesse	5
2	Hauptspeicher und Dateisysteme	63

Teil II: Rechnernetze

3	Anwendungen und Transport	101
4	Vermittlung und Übertragung	177
4.1	Aufgaben der Vermittlungsschicht	177
4.2	Prinzipien der Pfadermittlung	179
4.2.1	Globaler Link-State-Algorithmus	181
4.2.2	Dezentraler Distanzvektor-Algorithmus	184
4.2.3	Vergleich Link-State- versus Distanzvektor-Algorithmus	191
4.2.4	Hierarchisches Routing	194
4.3	Die Vermittlungsschicht im Internet	195
4.3.1	Das IP-Protokoll IPv4	196
4.3.2	Pfadbestimmung im Internet	202
4.3.3	Fehlerbehandlung und Abfragen im Internet	205
4.4	Neuere Entwicklungen für die Vermittlungsschicht	207
4.5	Aufgaben der Sicherungsschicht	208
4.6	Aufgaben der Bitübertragungsschicht	212
4.7	Vertiefungen	215
	Literatur	217

Die Autoren

Prof. Dr.-Ing. Anja Haake

Studium der Informatik an der Universität Dortmund zur Dipl.-Inform. 1988, dann Universität Dortmund und Institut für Integrierte Publikations- und Informationssysteme der GMD in Darmstadt, Promotion in Informatik 1996 an der Technischen Universität Darmstadt, danach Innovation Center der TLC GmbH und FernUniversität in Hagen, seit 2006 Professorin für Informatik und Wirtschaftsinformatik an der Fachhochschule Dortmund.

Univ.-Prof. Dr.-Ing. Jörg Haake

Studium der Informatik an der Universität Dortmund zum Dipl.-Inform. 1987, dann BCT GmbH und Institut für Integrierte Publikations- und Informationssysteme der GMD in Darmstadt, Promotion in Informatik 1995 an der Technischen Universität Darmstadt, Bereichsleiter am Fraunhofer Institut für Integrierte Publikations- und Informationssysteme (IPSI) in Darmstadt, seit 2001 Professor für Praktische Informatik an der FernUniversität in Hagen.

apl. Prof. Dr. rer. nat. Christian Icking

Studium der Mathematik und Informatik an der Universität Münster und der Université Pierre und Marie Curie Paris VI zur Maîtrise in Mathematik 1984, dann auch École Nationale Supérieure de Techniques Avancées Paris zum DEA (Diplom) Informatik 1985, danach Universität Karlsruhe, Universität Freiburg, Universität Essen und FernUniversität in Hagen, hier Promotion und Habilitation in Informatik 1994 und 2002, außerplanmäßiger Professor 2010.

Dr. rer. nat. Lihong Ma

Studium der Mathematik an der Universität Yunnan, Bachelor in Mathematik 1984, dann Technische Universität Kunming, Universität Karlsruhe, Universität Freiburg, dort Dipl.-Math. 1990, danach Universität Essen und FernUniversität in Hagen, hier Promotion in Informatik 2000, wissenschaftliche Mitarbeiterin.

Kurseinheit 4

Vermittlung und Übertragung

Diese Kurseinheit behandelt die tieferen Schichten des Schichtenmodells und Aspekte des Netzkerns. Wir behandeln die Prinzipien und Konzepte der Vermittlungsschicht (Abschnitt 4.1 bis 4.4) und geben einen kurzen Überblick über die Sicherungsschicht (Abschnitt 4.5). Um die Behandlung der Schichten des Protokollstapels abzuschließen, gehen wir in Abschnitt 4.6 kurz auf die unterste Schicht ein, die Bitübertragungsschicht. In Abschnitt 4.7 weisen wir interessierte Leser auf mögliche Vertiefungen des Themas Rechnernetze hin.

4.1 Aufgaben der Vermittlungsschicht

In Abschnitt 3.4 haben wir gesehen, wie die Transportschicht Kommunikationsdienste zwischen zwei Anwendungsprozessen bereitstellt, die auf zwei verschiedenen Endsystemen laufen. Hierbei benutzt die Transportschicht die Dienste der Vermittlungsschicht. Die Aufgabe der Vermittlungsschicht (network layer) ist der Transport von Transportschicht-Nachrichten, auch Segmente genannt, von einem sendenden Host zu einem empfangenden Host (*Host-zu-Host-Vermittlung*): Aus Sicht der Transportschicht kommunizieren die beiden Hosts, auf denen die beiden Anwendungsprozesse (Sender und Empfänger) laufen, über die Vermittlungsschicht explizit miteinander. Die Vermittlung eines Segments in der Vermittlungsschicht von Host zu Host umfasst jedoch alle Transitsysteme (Paket-Switches), die auf dem Pfad liegen, der die beiden Hosts verbindet. Dazu muss die Vermittlungsschicht die folgenden Teilaufgaben lösen:

- die *Pfadermittlung* (*Routing*): Hierbei muss die Vermittlungsschicht die Topologie der Router kennen und einen geeigneten Pfad bestimmen, den ein Paket, in dem ein Segment verpackt wurde, vom Senderhost zum Empfängerhost nehmen soll.
- Die *Vermittlung von Paketen*: Hier muss für Pakete, die bei einem Transitsystem ankommen, zuerst entschieden werden, über welche ausgehende Verbindung diese weitergesendet werden sollen.

Host-zu-Host-
Vermittlung

Pfadermittlung

Vermittlung von
Paketen

Call-Setup

- *Call-Setup*: in VC-Netzwerken (z. B. ATM oder X.25, siehe Abschnitt 3.2.2) müssen alle Transitsysteme auf einem Pfad bei Aufbau eines virtuellen Kanals richtig konfiguriert werden. Das Internet ist aber ein Datagramm-Netzwerk, d. h. ein Host muss jedes einzelne Paket mit der Adresse des Zielhosts versehen und es dann ins Netzwerk einspeisen. Also wird hier kein Call-Setup benötigt. Deshalb wird diese Teilaufgabe hier nicht weiter behandelt.

verbindungsloser
Best-Effort-
Dienst

Das Dienstmodell auf der Vermittlungsschicht beschreibt die Eigenschaften, die der Kommunikationsdienst der Vermittlungsschicht den nutzenden Prozessen bietet. Bei der Datagramm-Vermittlungsschicht des Internets wird ein *verbindungsloser Best-Effort-Dienst* zum Transport von Paketen zwischen Sender und Empfänger angeboten. Dieser Dienst weist die folgenden Eigenschaften auf:

1. Pakete können verloren gehen,
2. Pakete können in einer anderen Reihenfolge empfangen werden, als sie gesendet wurden,
3. über die Dauer des Transports eines Pakets bis zur Ankunft beim Empfänger wird keine Aussage gemacht,
4. es werden keine Zusagen über die Bandbreite zwischen Sender und Empfänger gemacht, und
5. es werden keine Informationen bzgl. einer Überlastung des Netzwerks an den Sender- oder Empfängerprozess weitergegeben.

In einem VC-Netzwerk (z. B. ATM oder X.25) wird ein verbindungsorientiertes Dienstmodell realisiert. Bei einem *verbindungsorientierten Dienst* können zusätzliche Leistungen bzw. Eigenschaften für den Transport von Paketen zugesichert werden. Beispiele hierfür sind bei ATM-Netzwerken vier ATM-Dienstmodelle, die bezüglich der oben erwähnten Eigenschaften weitergehende Garantien abgeben. So garantiert z. B. bei ATM das Constant-Bit-Rate-Dienstmodell eine angeforderte Bandbreite, den vollständigen Transfer sämtlicher Pakete in der richtigen Reihenfolge und die Aufrechterhaltung der zeitlichen Abstände zwischen den Paketen.

Übungsaufgabe 4.1 Warum kann das Internet nur einen Best-Effort-Dienst anbieten?

<https://e.feu.de/1801-best-effort>



In Abschnitt 4.2 behandeln wir zuerst die allgemeinen Grundprinzipien des Routing und danach in Abschnitt 4.3 als Konkretisierung die Vermittlungsschicht im Internet. Schließlich erwähnen wir in Abschnitt 4.4 noch die Erweiterungen, welche die neue Version IPv6 in der Vermittlungsschicht bietet.

4.2 Prinzipien der Pfadermittlung

In jedem Datagramm- oder VC-Netzwerk muss die Vermittlungsschicht den *Pfad* (die Route, siehe Abschnitt 3.2.2) für die Pakete festlegen, die diese vom Sender zum Empfänger nehmen sollen. Die Berechnung dieses Pfades ist die Aufgabe des Routing-Protokolls. Im *Routing-Protokoll* legt der *Routing-Algorithmus* fest, welchen Pfad ein Paket nehmen soll. Hierbei soll der Routing-Algorithmus einen *guten* Pfad aus der Menge aller möglichen Pfade zwischen dem Senderhost und dem Empfängerhost auswählen. Üblicherweise wendet man hier ein Kostenmodell an, das jedem Pfad Gesamtkosten zuweist. Der Algorithmus soll dann den billigsten Pfad berechnen, wobei in der Praxis auch noch weitere Kriterien zu berücksichtigen sind: z. B. sollen durch ein Firmennetzwerk keine Pakete aus den Netzwerken von konkurrierenden Firmen weitergeleitet werden.

Grundlage zur Formulierung von Routing-Algorithmen sind *gewichtete Graphen*, die das Netzwerk modellieren. Ein Graph besteht aus einer Menge von Knoten, die durch Kanten miteinander verbunden sind. In gewichteten Graphen ist zusätzlich jeder Kante eine Zahl zugeordnet, die ausdrückt, welche Kosten entstehen, wenn man diese Kante nutzt, um zum direkten Nachbarknoten zu gelangen. Bei der Nutzung von Graphen für die Formulierung von Routing-Algorithmen stellen die Knoten die Transitsysteme dar, in denen die Routing-Entscheidungen getroffen werden. Die Kanten repräsentieren die Kommunikationsverbindungen zwischen den Transitsystemen. Die Gewichte an den Kanten geben die Kosten der Nutzung dieser Verbindung an, z. B. die aktuelle Wartezeit bei Nutzung der Verbindung oder die Länge der Verbindung (interkontinentale Verbindungen können teurer als lokale Verbindungen sein).

Die Aufgabe eines Routing-Algorithmus ist nun die Ermittlung des billigsten Pfades zwischen einem Quellknoten und einem Zielknoten. In der Literatur wird diese Aufgabe auch oft als *Wegauswahl* bezeichnet, man spricht dann auch von Wegauswahlverfahren. Die Kosten eines Pfades sind die Summe der Kosten der einzelnen Verbindungen auf dem Pfad. Die billigsten Kosten von einem Quellknoten zu einem Zielknoten sind das Minimum aller Kosten über alle möglichen Pfade zwischen der Quelle und dem Ziel. Abbildung 4.1 zeigt ein Beispiel für die Modellierung eines Netzwerks aus 6 Transitsystemen von *A* bis *F* und 10 Kommunikationsverbindungen. Der billigste Pfad zwischen *A* und *C* hat die Kosten 3 und führt von *A* über *D* und *E* zu *C*.

Pfad

Routing-
Protokollgewichtete
Graphen

Wegauswahl

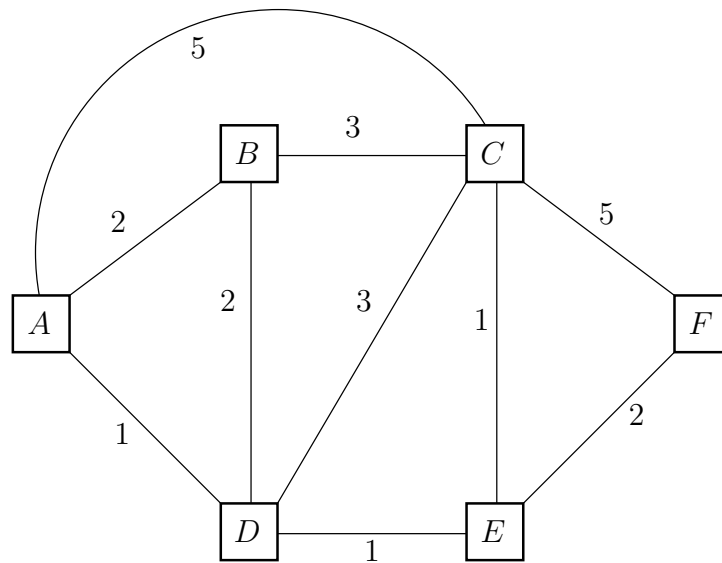


Abbildung 4.1: Modellierung eines simplen Netzwerks.



Übungsaufgabe 4.2 Wie lautet der billigste Pfad von A zu F in Abbildung 4.1?

<https://e.feu.de/1801-billigster-pfad-1>



Wie haben Sie diesen Pfad ermittelt? Meistens probiert man ein paar Pfade von A nach F aus, und wählt dann den billigsten. Aber haben Sie auch alle 17 möglichen Wege systematisch ausprobiert? Dabei ist diese Berechnung ein Beispiel für einen *zentralisierten Routing-Algorithmus*: der Algorithmus wurde an einer Stelle durchgeführt, nämlich durch Sie, und hatte komplette Information über das Netzwerk, nämlich Abbildung 4.1.

Wir können Routing-Algorithmen anhand der folgenden Unterscheidungsmerkmale klassifizieren:

- Global – dezentral:

- Ein *globaler Routing-Algorithmus* besitzt vollständiges Wissen über das Netzwerk. Er kennt also alle Verbindungen und alle Kosten. Diese Informationen muss der Routing-Algorithmus irgendwie sammeln, bevor ein Pfad berechnet werden kann. Die eigentliche Berechnung kann sowohl zentral an einer Stelle als auch verteilt an mehreren Stellen stattfinden. Solche Algorithmen mit globalen Zustandinformationen heißen deshalb *Link-State-Algorithmen*.
- Ein *dezentraler Routing-Algorithmus* berechnet einen Pfad mit einem iterativen, verteilten Verfahren, ohne dass jeder Knoten die Verbindungskosten des gesamten Netzwerks kennen muss. Stattdessen kennt jeder Knoten nur die Informationen über seine direkten

global versus
dezentral

Verbindungen. Mittels Informationsaustausch mit seinen Nachbarn im Graphen kann der Knoten sukzessive den billigsten Pfad zu einem Zielknoten berechnen. Ein Beispiel für solch einen Algorithmus ist der *Distanzvektor-Algorithmus*, bei dem jeder Knoten nur weiß, zu welchem Nachbarn ein Paket weitergeleitet werden muss, um einen speziellen Zielknoten zu erreichen, und was die Kosten des Pfades von ihm selbst zum Zielknoten sind.

- Statisch – dynamisch:

- Ein *statischer Routing-Algorithmus* ändert seine Routing-Entscheidungen nur aufgrund manueller Eingaben, z. B. dem Editieren von Routing-Tabellen in Routern. Dadurch kann ein statischer Routing-Algorithmus nur langsam an veränderte Netzwerksituationen angepasst werden, z. B. bei Ausfall von Verbindungen.
- Ein *dynamischer Routing-Algorithmus* passt sich automatisch an veränderte Netzwerksituationen an, wie z. B. Netzwerkauslastung oder Ausfall von Verbindungen. Die Anpassung kann hierbei entweder periodisch durchgeführt, oder durch Zustandsveränderungen im Netzwerk ausgelöst werden.

statisch versus
dynamisch

Im Internet sind nur zwei Verfahren gebräuchlich: ein dynamischer globaler Link-State-Algorithmus, siehe Abschnitt 4.2.1, und ein dynamischer dezentraler Distanzvektor-Algorithmus, siehe Abschnitt 4.2.2. Routing-Algorithmen für andere Netzwerktypen werden hier nicht weiter besprochen. Sehen Sie hierzu bei Bedarf in der Fachliteratur nach, z. B. Kapitel 4 bei Kurose und Ross [9].

4.2.1 Globaler Link-State-Algorithmus

Voraussetzung für den Algorithmus ist, dass *alle* Knoten *alle* Informationen über die Netzwerktopologie und die Verbindungskosten vorliegen haben. Dies wird in der Praxis dadurch erreicht, dass jeder Knoten die Informationen über die Kosten der an ihn angeschlossenen Verbindungen in sogenannten *Link-State-Broadcast-Nachrichten* an alle anderen Knoten verschickt (Broadcast). Wenn alle Knoten alle ihre Link-State-Broadcast-Nachrichten verschickt haben, dann hat jeder Knoten die komplette Information über die Netzwerktopologie und die Verbindungskosten vorliegen. Damit kann nun jeder Knoten die billigsten Pfade zu allen anderen Knoten berechnen.

Link-State-
Broadcast-
Nachrichten

Dijkstras iterativer Algorithmus berechnet den billigsten Pfad von *einem* gegebenen Knoten (der Quelle Q) zu *allen* anderen Knoten im Netzwerk. Nach der k -ten Iteration wurden die billigsten Pfade zu k Zielknoten berechnet. Diese k Pfade haben die k billigsten Kosten in der Menge aller billigsten Pfade zu den k Zielknoten. Es gelte die folgende Notation:

Dijkstras
Algorithmus

- $c(i, j) \geq 0$ bezeichnet die Kosten der direkten Verbindung von Knoten i zu Knoten j . Falls keine direkte Verbindung existiert, dann ist $c(i, j) = \infty$. Außerdem setzen wir die Symmetrie-Eigenschaft voraus, d. h. $c(i, j) = c(j, i)$.
- d_i bezeichnet die Kosten des Pfades vom Quellknoten Q zum Zielknoten i , der in dieser Iteration die billigsten Kosten hat.
- P_i bezeichnet den Vorgängerknoten (einen Nachbarn von Knoten i) auf dem gegenwärtig billigsten Pfad vom Quellknoten zum Zielknoten i .
- S bezeichnet die Menge der Knoten, bei denen die billigsten Pfade vom Quellknoten bereits bekannt sind.

Der Algorithmus besteht aus einer Initialisierung und der Iteration (d. h. einer Schleife). Nach der Beendigung des Algorithmus sind die billigsten Pfade vom Quellknoten zu allen anderen Knoten bekannt.

- Initialisierung:
 $S = \{Q\}$ /* Q ist der Quellknoten, an dem die Berechnung ausgeführt wird */
Für alle Knoten i des Netzwerkes
begin
Falls i direkter Nachbar von Q ist
dann $d_i = c(Q, i)$; $P_i = Q$;
sonst $d_i = \infty$.
end
- Iteration:
Wiederhole solange, bis alle Knoten des Netzwerkes in S sind
begin
Finde ein $i \notin S$ mit $d_i = \min_{j \notin S} \{d_j\}$
Füge i zu S hinzu;
Update d_j für alle direkten Nachbarn $j \notin S$ von i
Falls $d_i + c(i, j) < d_j$ ist
dann $d_j = d_i + c(i, j)$; $P_j = i$;
/* die neuen Kosten von j sind entweder die alten Kosten oder die bekannten billigsten Kosten nach i plus den Kosten von i nach j */
end

Wir betrachten nun das Netzwerk in Abbildung 4.1 und berechnen die billigsten Pfade vom Quellknoten A zu allen möglichen Zielknoten.¹ Die Ergebnisse für jeden Iterationsschritt sind in Tabelle 4.1 dargestellt:

¹ P_X bezeichnet dann den Vorgänger von X mit $X \in \{B, C, D, E, F\}$.

Schritt	S	B, P_B	C, P_C	D, P_D	E, P_E	F, P_F
1	A	2, A	5, A	1, A	∞	∞
2	A, D	2, A	4, D		2, D	∞
3	A, D, E	2, A	3, E			4, E
4	A, D, E, B		3, E			4, E
5	A, D, E, B, C					4, E
6	A, D, E, B, C, F					

Tabelle 4.1: Zwischenergebnisse des Dijkstra-Algorithmus für Abbildung 4.1.

Bei Terminierung des Algorithmus liegt für jeden Knoten sein Vorgänger auf dem billigsten Pfad vom Quellknoten vor. Für jeden Vorgänger kennen wir außerdem dessen Vorgänger. So können wir die gesamten Pfade vom Quellknoten zu allen Zielknoten rekonstruieren, wir erhalten jetzt einen Graphen ohne geschlossene Pfade (Zyklen), siehe Abbildung 4.2. So einen Graphen bezeichnen wir als Baum.

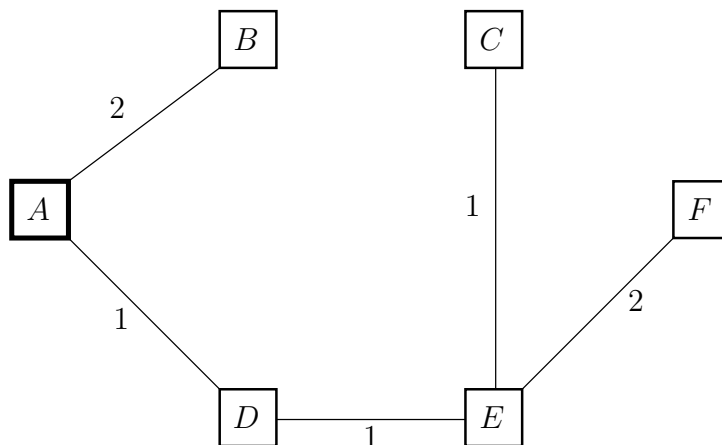


Abbildung 4.2: Die kürzesten Pfade von A zu allen Knoten bilden einen Baum mit der Wurzel A, ein Teilgraph des Netzwerks aus Abbildung 4.1.

Die Zeitkomplexität² dieses Algorithmus ist $O(n^2)$ für n Knoten, um den billigsten Pfad von einem Quellknoten zu allen $n - 1$ anderen Knoten zu berechnen. Der Algorithmus muss von außen neu angestoßen werden, um neue Pfade zu berechnen, wenn sich zum Beispiel die Auslastung (Kosten) von Verbindungen oder die Netzwerktopologie geändert hat.

²Wenn Sie die O -Notation noch nicht kennen: Im Allgemeinen steht $O(f(n))$ für eine Funktion, die höchstens proportional zu $f(n)$ wächst.

dezentrales,
iteratives,
asynchrones
Verfahren

Distanztabelle

4.2.2 Dezentraler Distanzvektor-Algorithmus

Der *Distanzvektor-Algorithmus* ist ein *dezentrales*, *iteratives* und *asynchrones* Verfahren zur Berechnung der billigsten Pfade:

- *Dezentral*: Jeder Knoten empfängt Informationen von seinen direkten Nachbarn, führt eine Berechnung durch und sendet gegebenenfalls das Ergebnis an seine direkten Nachbarn zurück.
- *Iterativ*: Das Verfahren läuft so lange, wie noch Informationen zwischen Nachbarn ausgetauscht werden müssen.
- *Asynchron*: Die Knoten tauschen Informationen aus, wenn sie es müssen, d. h. wenn sich bei ihnen eine Änderung der billigsten Pfade ergibt. Andere Knoten müssen auf diese Informationen nicht innerhalb einer bestimmten Zeit reagieren.

Grundlegende Datenstruktur des Algorithmus ist die *Distanztabelle*, die an jedem Knoten gespeichert ist. Die Distanztabelle enthält eine Zeile für jeden Zielknoten und eine Spalte für jeden seiner direkten Nachbarn. Jede Zelle $d_X(Z, N)$ der Matrix gibt an, zu welchen Kosten der Zielknoten Z vom aktuellen Knoten X über den Nachbarknoten N erreichbar ist. Diese Kosten entsprechen den Kosten $c(X, N)$ der direkten Verbindung von Knoten X zum Nachbarknoten N plus den billigsten Kosten über einen Pfad vom Nachbarknoten N zum Zielknoten Z , also

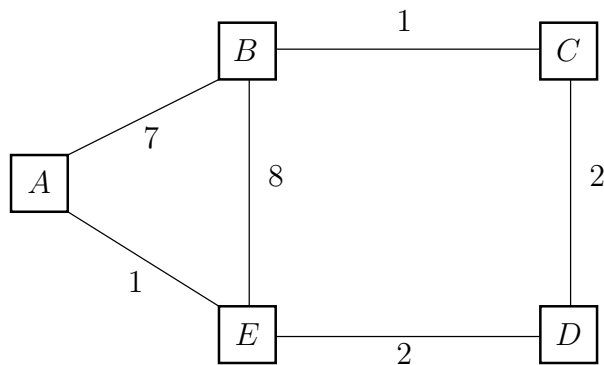
$$d_X(Z, N) = c(X, N) + \min_W \{d_N(Z, W)\}$$

wobei das Minimum über alle Nachbarknoten W von N genommen wird.

Jeder Knoten soll die *Kosten des billigsten Pfades* jedes seiner Nachbarn zu jedem Ziel kennen. Trägt ein Knoten neue billigste Kosten in seiner Distanztabelle ein, so muss er deshalb seine direkten Nachbarn darüber informieren.

Ein Beispiel soll die Bedeutung der Distanztabelle verdeutlichen. In Abbildung 4.3 ist ein Netzwerkmodell und die dazugehörige Distanztabelle für den Knoten E angegeben. Betrachten wir zuerst die Zeile für Zielknoten A :

- Der billigste Pfad von E nach A geht über die direkte Verbindung zu A : die Kosten sind 1. Deshalb ist $d_E(A, A) = 1$.
- Die Kosten des billigsten Pfades von E nach A direkt über den Nachbarknoten D ergibt sich zu 5: die Kosten von E nach D sind 2 plus den billigsten Weg von D nach A (nämlich zurück zu E und weiter zu A) mit den Kosten 3. Die *Schleife*, die denselben Knoten (hier E) mehrfach benutzt, ist die Quelle einiger Probleme, wie wir noch sehen werden.
- Der billigste Pfad von E nach A direkt über den Nachbarknoten B ergibt sich zu 14.

Distanztabelle für E

		Nachbar		
$d_E()$		A	B	D
Ziel	A	<u>1</u>	14	5
	B	7	8	<u>5</u>
	C	6	9	<u>4</u>
	D	4	11	<u>2</u>

Abbildung 4.3: Die Distanztabelle für Knoten E . Die billigsten Kosten zum entsprechenden Ziel über einen entsprechenden Nachbarn sind fett und unterstrichen dargestellt.

Übungsaufgabe 4.3 Warum ist 14 das Ergebnis für den billigsten Pfad von E nach A direkt über den Nachbarn B in Abbildung 4.3 und nicht $15 = 8 + 7$?

<https://e.feu.de/1801-billigster-pfad-2>



In der Distanztabelle in Abbildung 4.3 sind die billigsten Pfade von E zu jedem Zielknoten fett markiert.

Aus seiner Distanztabelle kann also jeder Knoten seine *Routing-Tabelle* erstellen, d. h. die Minima jeder Zeile. Diese Tabelle gibt an, über welchen Nachbarn die Pakete an ein bestimmtes Ziel weitergeleitet werden sollen.

Damit die Routing-Tabellen verteilt berechnet werden können, also ohne Kenntnis des gesamten Graphen, tauschen beim Distanzvektor-Algorithmus nur die direkten Nachbarn Informationen aus. Hierzu benutzen sie den sogenannten *verteilten Bellman-Ford-Algorithmus*, der an jedem Knoten X ausgeführt wird. Er besteht aus einer Initialisierungsphase und einer Iterationsphase.

Routing-Tabelle

verteilter
Bellman-Ford-
Algorithmus

Initialisierung:

```

01 Für alle benachbarten Knoten  $V$  tue:
02 begin
03    $d_X(*, V) = \infty$  /* der *-Operator bedeutet für alle Zeilen */
04    $d_X(V, V) = c(X, V)$ 
05 end
06 Für alle Zielknoten  $Z$  tue:
07 begin
08   Sende  $\min_W \{d_X(Z, W)\}$  an jeden direkten Nachbarn
09 end
  
```

Iterationsphase:

```

01 Wiederhole für immer
02 begin
03     Warte, bis entweder eine Kostenveränderung der Verbindung zum
04     direkten Nachbarn  $V$  bemerkt wird oder bis ein Update der
05     Verbindungskosten vom Nachbarn  $V$  eintrifft.

06     Falls  $c(X, V)$  sich um die Differenz  $\delta$  geändert hat
07     begin
08         Für alle Zielknoten  $Z$  tue:
09         begin
10             /* addiere die positive oder negative Differenz zu allen
11             Kosten der Wege, die über  $V$  führen */
12              $d_X(Z, V) = d_X(Z, V) + \delta$ 
13         end
14     end
15     Sonst, falls ein Update von Nachbar  $V$  bezüglich Zielknoten  $Z$ 
16     empfangen wird
17     /* d. h. der billigste Pfad von  $V$  nach  $Z$  hat sich geändert:  $V$  hat
18     einen neuen Wert  $neuerWert = \min_W \{d_V(Z, W)\}$  geschickt. */
19     begin
20          $d_X(Z, V) = c(X, V) + neuerWert$ 
21     end

22     Falls ein neues  $\min_W \{d_X(Z, W)\}$  für einen Zielknoten  $Z$  existiert
23     begin
24         sende den neuen minimalen Kostenwert  $\min_W \{d_X(Z, W)\}$  für  $Z$ 
25         an alle direkten Nachbarn.
26     end
27 end

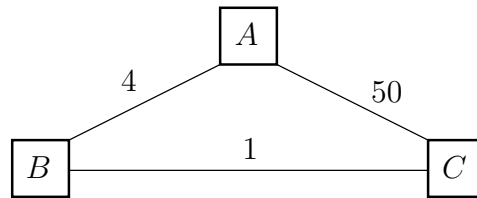
```

Der Knoten X verändert seine Distanztabelle, wenn er eine Kostenveränderung einer seiner abgehenden Verbindungen bemerkt (Zeilen 6-14) oder wenn er ein Update von einem direkten Nachbarn erhält (Zeilen 15-21). Wenn ein neuer billigster Pfad ermittelt wurde, schickt der Knoten diesen Wert an alle direkten Nachbarn (Zeilen 22-26). Der aktuelle Knoten kennt die Werte $\min_W \{d_V(Z, W)\}$ ausschließlich aufgrund der Updates, die er vom Nachbarknoten v erhalten hat. Sobald keine neuen minimalen Kosten mehr ermittelt werden, werden auch keine Update-Nachrichten mehr verschickt: jeder Knoten wartet bei der Warte-Anweisung in der Schleife (Zeilen 3-5), bis sich die Kosten einer Verbindung ändern.

Wenn sich Verbindungskosten ändern, dann aktualisiert der Knoten seine Distanztabelle und benachrichtigt seine Nachbarn, falls ein neuer billigster Pfad berechnet wurde. Diese Propagation von neuen Werten kommt üblicherweise relativ rasch zum Stillstand. Abbildung 4.4 zeigt ein einfaches Netzwerk und den Aufbau der zugehörigen Routing-Tabellen, der bereits nach

Konvergenz

der zweiten Iteration konvergiert: Nach dem ersten Austausch von Routing-Informationen ändern sich nur noch die Werte für d_B ,³ woraus sich aber keine neuen Minima-Informationen ergeben, die weitergeschickt werden müssten.



Distanz-
Tabellen

	Knoten A			Knoten B			Knoten C		
		B	C		A	C		A	B
(a)	B	<u>4</u>	∞	A	<u>4</u>	∞	A	<u>50</u>	∞
	C	∞	<u>50</u>	C	∞	<u>1</u>	B	∞	<u>1</u>

		B	C		A	C		A	B
(b)	B	<u>4</u>	51	A	<u>4</u>	51	A	50	<u>5</u>
	C	<u>5</u>	50	C	54	<u>1</u>	B	54	<u>1</u>

		B	C		A	C		A	B
(c)	B	<u>4</u>	51	A	<u>4</u>	6	A	50	<u>5</u>
	C	<u>5</u>	50	C	9	<u>1</u>	B	54	<u>1</u>

Abbildung 4.4: Verteilter Bellman-Ford-Algorithmus auf einem Netzwerk:

- (a) Distanztabelle nach der Initialisierung
- (b) nach der ersten Iteration
- (c) nach der zweiten Iteration

Übungsaufgabe 4.4 Wir betrachten den Graphen in Abbildung 4.3 und wollen einen Schritt des dezentralen *Distanzvektor-Algorithmus* ablaufen lassen. Knoten A und B haben ihre Distanztabelle schon wie folgt initialisiert:

Knoten A			Knoten B			
$d_A()$	B	E	$d_B()$	A	C	E
B	7	∞	A	7	∞	∞
E	∞	1	C	∞	1	∞
			E	∞	∞	8

³Der Weg von B nach A über C ergibt aus der Information, dass C die minimalen Kosten 5 hat, um nach A zu kommen und B weiß, dass er die minimalen Kosten 1 hat, um nach C zu kommen. Der Weg von B über A nach C ergibt sich daraus, dass A die minimalen Kosten 5 hat, um nach C zu kommen und B weiß, dass er minimale Kosten von 4 hat, um nach A zu kommen.



Wie sehen die aktualisierten Distanztabellen aus, nachdem Knoten B und A ihre Informationen ausgetauscht haben? Bitte füllen Sie die vorgegebenen Tabellen aus:


Knoten A

$d_A()$	B	E
B		
C		
E		

Knoten B

$d_B()$	A	C	E
A			
C			
E			

<https://e.feu.de/1801-distanztabelle>



Routing-Schleife

Ein typisches Problem kann auftreten, wenn sich die Kosten einer Verbindung erhöhen und sich dadurch eine Routing-Schleife bildet, siehe Abbildung 4.5: Wir nehmen an, dass sich die Verbindungskosten der Verbindung von B nach A plötzlich auf 60 erhöhen, siehe Abbildung 4.5 (1). B erkennt nun, dass der Pfad nach A über C billiger ist, nämlich 6, als der direkte Weg zu A . Gemäß des verteilten Bellman-Ford-Algorithmus informiert B nun C über dieses neue Minimum. Knoten C wird seine Kosten für den Weg zu A über B jedoch nur auf 7 erhöhen, siehe Abbildung 4.5 (1): Aus der Sicht von C gelangt man am besten zu A über B , indem man direkt zu B gelangt, also 1, plus die Kosten, die B braucht, um zu A zu gelangen, nämlich 6. Knoten C schickt also weiterhin Pakete an A über den Knoten B , da ja die Verbindungskosten von 7 immer noch billiger sind als die direkte Verbindung von C nach A in Höhe von 50. In diesem Zustand der Routing-Tabellen, siehe Abbildung 4.5 (1), werden die Pakete von C nach A endlos zwischen C und B hin und her geschickt. Diese Situation bezeichnet man auch als *Routing-Schleife*. Solange die Routing-Tabellen nicht die global richtige Situation widerspiegeln, werden die Pakete niemals ihr Ziel erreichen! Die Situation verbessert sich erst schrittweise durch die Aktualisierung der Tabellen gemäß des verteilten Bellman-Ford-Algorithmus.

Tatsächlich erhöhen sich die Verbindungskosten in unserem Beispiel nur jeweils um den Wert 1 bei jedem Update, da jeweils nur die Kosten der direkten Verbindung zwischen C und B in Höhe von 1 auf die Kosten der Verbindung von C nach A addiert werden, siehe Abbildung 4.5 (2), (3). Insgesamt müssen 45 Updates zwischen B und C ausgetauscht werden, bevor C letztendlich feststellt, dass die Verbindung zu A über B teurer ist als seine direkte Verbindung zu A . Im Allgemeinen gilt: erhöhen sich die Kosten einer Verbindung, so wird diese schlechte Neuigkeit erst langsam durch das Netzwerk propagiert. Die anderen Knoten werden solange noch Pakete weiter über die nun schlechte (da teure) Verbindung schicken, bis bei ihnen neue Werte angekommen sind, die schlechter als andere alternative Pfade sind. Im Extremfall erhöht sich

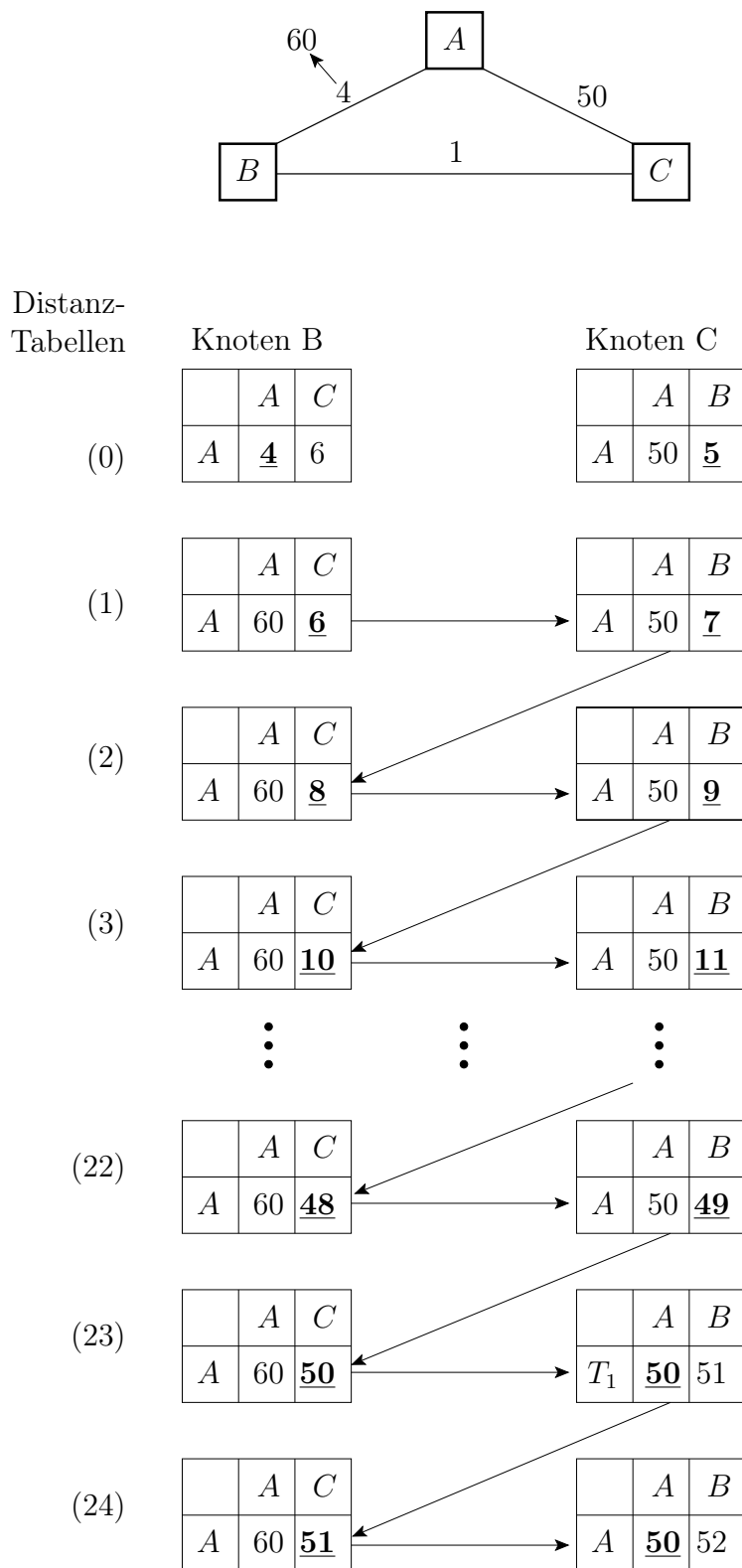


Abbildung 4.5: Das Entstehen einer Routing-Schleife und das Count-to-Infinity-Problem.

Count-to-Infinity-
Problem

der propagierte neue Wert pro Iteration aber nur um 1, siehe Abbildung 4.5. Es braucht also entsprechend viele Iterationen, bis ein anderer, kostengünstigerer Pfad berechnet wird. Bis dahin wird der Verkehr noch über die schlechte Verbindung abgewickelt. Dieses Problem heißt in der Literatur auch *Count-to-Infinity-Problem*.

Poisoned-
Reverse-Strategie

Um das Problem der Routing-Schleife zu vermeiden, kann die sogenannte *Poisoned-Reverse-Strategie* angewendet werden: Wenn der Knoten *C* Pakete zum Zielknoten *A* über einen anderen Knoten, hier *B*, leitet, obwohl er selbst eine direkte Verbindung zum Zielknoten hat, dann teilt er *B* einfach mit, dass die Kosten der direkten Verbindung von *C* zum Zielknoten *A* bei ∞ liegen, siehe Abbildung 4.6 (1). So wird *B* niemals Nachrichten an *A* über den Kno-

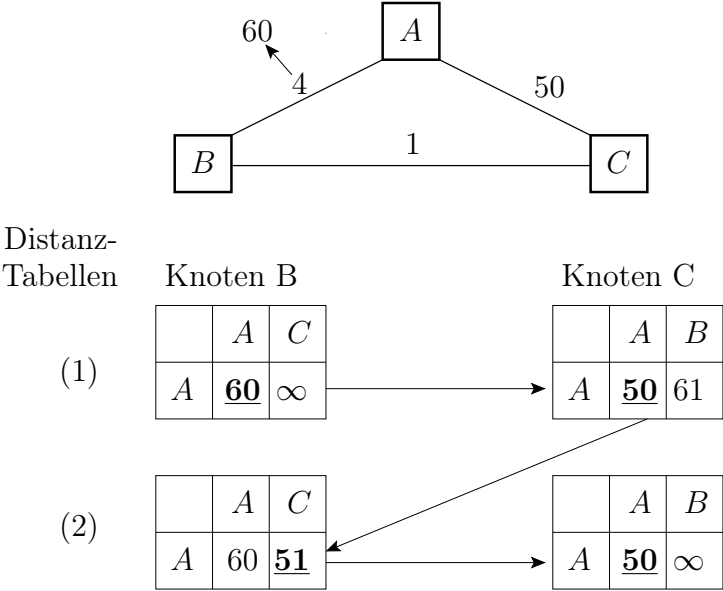


Abbildung 4.6: Knoten *C* und *B* wenden die Poisoned-Reverse-Strategie an.

ten *C* schicken! Sobald die Verbindungskosten über *B* teurer werden als die Kosten der direkten Verbindung zum Zielknoten, zum Beispiel wenn der Verbindung von *B* nach *A* plötzlich die Kosten 60 zugewiesen werden, dann wird der Knoten *C* die wirklichen Kosten seiner Verbindung zu *A* an *B* schicken, nämlich hier den Wert 50, siehe Abbildung 4.6 (1). Knoten *B* wird dann seinen Verkehr zum Zielknoten *A* über den Knoten *C* abwickeln, da *A* dann über *C* zu den Kosten von 51 erreichbar ist und das billiger als die eigenen direkten Verbindungskosten von *B* nach *A* von 60 ist, siehe Abbildung 4.6 (2). Schleifen, die sich aus mehr als zwei Knoten zusammensetzen, werden durch diese Strategie jedoch nicht bemerkt. Zudem löst diese Strategie auch nicht immer das Count-to-Infinity-Problem. Dies wird an folgendem Beispiel deutlich:

Beispiel

Ein Netzwerk besteht aus vier Knoten A , B , C und D und den Verbindungen $A-B$, $B-C$ und $C-D$ mit Kosten von jeweils 1 und $A-D$ mit Kosten 50, siehe auch Abbildung 4.7.

Die Knoten kennen die folgenden Informationen über die Kosten, um zu Knoten A zu gelangen:

- Knoten B erreicht A direkt mit den minimalen Kosten 1.
- Knoten C erreicht A über B mit den minimalen Kosten 2.
- Knoten D erreicht A über C mit den minimalen Kosten 3.
- Knoten D verwendet nun die Poisoned-Reverse-Strategie und teilt C mit, dass A über ihn mit den Kosten ∞ erreicht wird, da D weiß, dass er A mit Kosten 3 über C viel billiger als die direkte Verbindung erreichen kann.

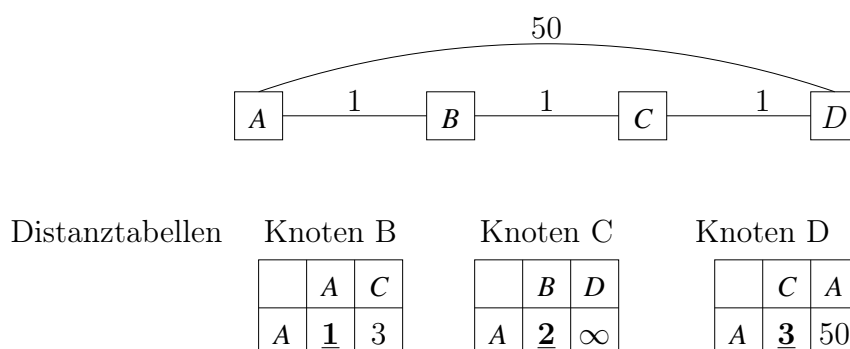


Abbildung 4.7: Die Distanztabelle von Knoten B , C und D , um Zielknoten A zu erreichen. Knoten D verwendet die Poisoned-Reverse-Strategie.

Wenn nun die Verbindung $A-B$ unterbrochen wird, siehe Abbildung 4.8, dann teilt Knoten B Knoten C mit, dass A über ihn mit Kosten 3 erreichbar ist, siehe Schritt (1) in Abbildung 4.8. C aktualisiert die Kosten nach A mit dem neuen Wert 4 in Schritt (2). Knoten C teilt B und D mit, dass A über ihn mit Kosten 4 erreichbar ist. Knoten B und D erhöhen die Kosten nach A jeweils in Schritt (3) um 1. Jetzt haben wir das Count-to-Infinity-Problem.

4.2.3 Vergleich Link-State- versus Distanzvektor-Algorithmus

Wir vergleichen die beiden Algorithmen bezüglich ihrer Komplexität, Konvergenzgeschwindigkeit, d. h. der Geschwindigkeit, in der die Algorithmen zu einer stabilen Lösung führen, und Robustheit. Dabei gehen wir von einem Netzwerk mit n Knoten und L Verbindungen aus.

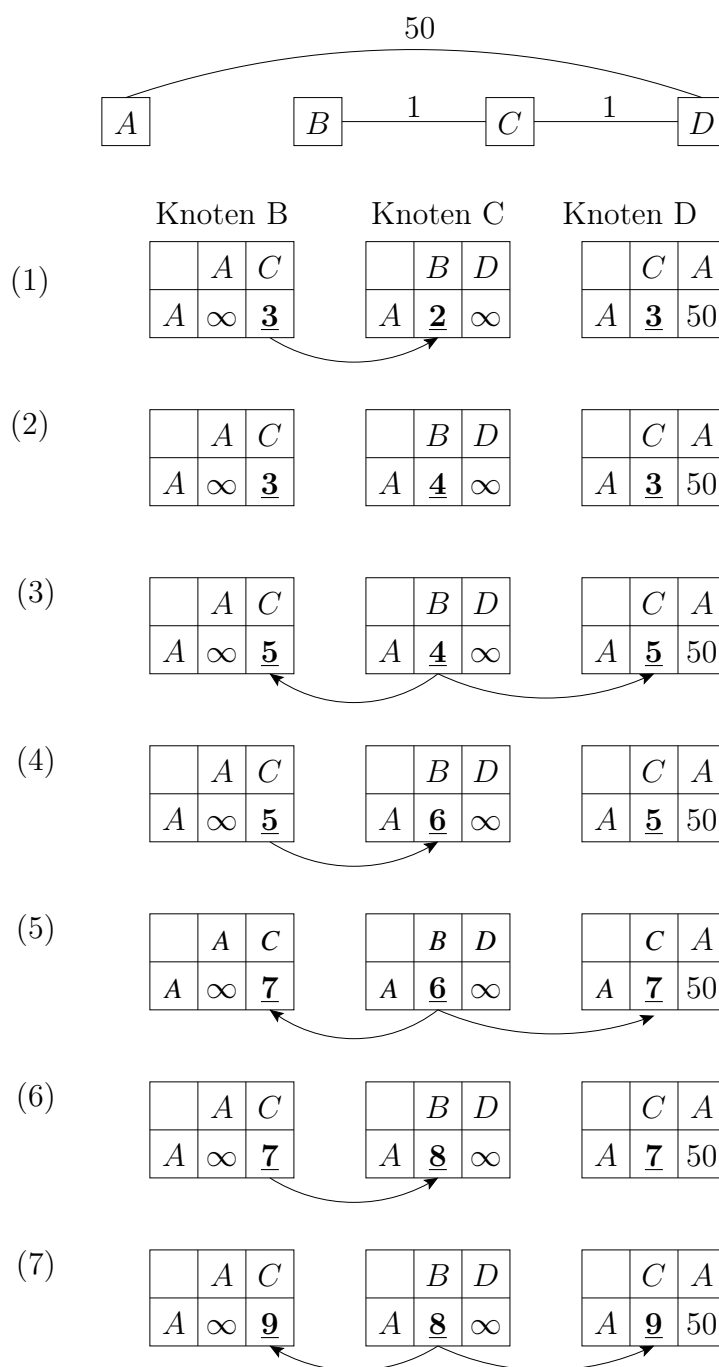


Abbildung 4.8: Ein Count-to-Infinity-Problem entsteht, obwohl Knoten D die Poisoned-Reverse-Strategie verwendet.

- Komplexität bzgl. Nachrichtenmenge:
 - Link-State-Algorithmus: Das Senden von $O(nL)$ Nachrichten sind notwendig, damit jeder Knoten den gesamten Graphen mit Kosten kennt. Bei der Verbreitung der Nachricht wird die *Fluten-Methode* verwendet. Fluten funktioniert wie folgt: Zuerst wird ein Nachricht-Paket mit der Identität (IP-Adresse) des Knotens, mit Kosten zu di-

rekten Nachbarn und mit einer Sequenznummer für das Paket an die Nachbarn gesendet. Die Übertragung kann zuverlässig gemacht werden, indem der Sender und der Empfänger den Bestätigungs- und Neuübertragungsmechanismus verwenden. Wenn ein weiterer Knoten das Paket empfängt, speichert er das Paket, falls er noch nie so ein Paket mit der Identität eines Knotens erhalten hat. Verfügt er bereits über eine Kopie, vergleicht er die beiden Sequenznummern. Nur wenn das neue Paket eine größere Sequenznummer besitzt, ersetzt er die Kopie durch das neue Paket, da er vermutet, dass das neu angekommene Paket wirklich neu sein muss. Ist das empfangene Paket das neue, sendet der Knoten jetzt eine Kopie an alle seine Nachbarn, außer an den, von dem er das Paket vorher empfangen hat. Also wird ein Link-State-Paket nie zu dem Knoten zurückgesendet, von dem es empfangen wurde. Das garantiert, dass das Fluten terminiert.

- Distanzvektor-Algorithmus: In jeder Iteration müssen Nachrichten nur dann geschickt werden, wenn sich ein anderer billigster Weg zu einem Knoten ergibt, und zwar nur an die direkten Nachbarn.
- Konvergenzgeschwindigkeit:
 - Link-State-Algorithmus: Bei jedem Knoten wird der Dijkstra-Algorithmus mit $O(n^2)$ Operationen ausgeführt, um den billigsten Pfad von ihm zu allen $n - 1$ anderen Knoten zu berechnen. Der Algorithmus terminiert und hat eine Laufzeit $O(n^2)$. Insbesondere berechnet jeder Knoten seine Routing-Tabelle mit Dijkstra-Algorithmus einmal, dann bleibt die Tabelle stabil erhalten, bis die nächste Veränderung passiert. Es gibt beim Linkstate-Algorithmus keine Konvergenzphase.
 - Distanzvektor-Algorithmus: Beim Distanzvektor-Algorithmus findet eine Konvergenzphase statt, in der Updates immer wieder durchgeführt werden müssen. Während der Konvergenzphase bleibt die Routing-Tabelle nicht stabil und es ist nicht zu sehen, wie lange die Veränderung der Tabelle dauert. Während der Konvergenzphase können Routing-Schleifen entstehen, die durch das Count-to-Infinity-Problem verursacht werden.
Der Algorithmus reagiert schnell auf gute Nachrichten und konvergiert aber langsam bei schlechten Nachrichten.
- Robustheit: Wenn ein Knoten ausfällt oder fehlerhaft ist, dann werden seine Nachbarknoten es merken.
 - Link-State-Algorithmus: Jeder Nachbar wird eine neue Link-State-Nachricht im ganzen Netz verbreiten, d. h. dieser fehlerhafte Knoten existiert nicht mehr im Netz. Danach berechnet jeder Knoten seine eigene neue Routing-Tabelle. Hierdurch entsteht ein gewisses Ausmaß von Robustheit des verteilten Systems. Der Algorithmus wird in der Praxis recht häufig verwendet.

- Distanzvektor-Algorithmus: Jeder Nachbarknoten des ausgefallenen Knotens wird seine Routing-Tabelle aktualisieren und die neuen Kosten seinen Nachbarn mitteilen, die wiederum ihre Routing-Tabellen aktualisieren und die neuen Kosten an ihre Nachbarn weitergeben. In dieser Zeit können Routing-Schleifen entstehen, bei denen Pakete im Kreis herum geschickt werden, dadurch kann das ganze Netz zusammenbrechen.

Der Link-State-Algorithmus ist besser als der Distanzvektor-Algorithmus, wenn man nur die Komplexität, Robustheit und Konvergenzgeschwindigkeit betrachtet. Leider kann man so einen Algorithmus nicht für ein großes Netzwerk verwenden, da jeder Knoten den gesamten Graphen des Netzwerks kennen und speichern muss. Wenn man einen Graphen als Adjazenzmatrix speichert, dann benötigt jeder Knoten $O(n^2)$ Speicherplatz. Wenn man einen Graphen in Form von Adjazenzenlisten speichert, d.h. jeder Knoten im Graphen hat eine Liste von Nachbarn, dann hat jeder Knoten $O(nL)$ Speicherbedarf.

4.2.4 Hierarchisches Routing

In der bisherigen Diskussion haben wir das Netzwerk als Menge verbundener Transitsysteme angesehen. Jedes Transitsystem wendet denselben Routing-Algorithmus an, um Pfade durch das gesamte Netzwerk zu berechnen. In der Praxis funktioniert diese Sicht aus zwei Gründen nicht:

Größe realer
Netzwerke

1. Aufgrund der schieren *Größe realer Netzwerke*, z. B. des Internets (Millionen von Routern), ist der Aufwand für die Berechnung, Speicherung und Weiterleitung von Routing-Tabellen-Informationen nicht mehr tragbar. Ein Distanzvektor-Algorithmus mit Iterationen durch Millionen von Routern würde wohl kaum konvergieren. Deshalb muss die Komplexität der Pfadberechnung in großen Netzwerken reduziert werden.

heterogene
Routing-
Anforderungen

2. Organisationen wollen ihre Netzwerke so betreiben, wie sie es wollen. Sie wollen z. B. einen Routing-Algorithmus auswählen, der auf ihre Bedürfnisse am besten passt. Trotzdem sollte das Netzwerk mit anderen *außenstehenden* Netzwerken verbunden werden können.

AS

Beide Probleme können durch das Konzept der autonomen Systeme gelöst werden. In einem *autonomen System* (AS) wird eine Menge von Transitsystemen zusammengefasst, die alle denselben Routing-Algorithmus benutzen, der auch *Intra-AS-Routing* genannt wird. Sogenannte *Gateways* oder *Gateway-Router* verbinden verschiedene autonome Systeme. Gateways sind spezielle Transitsysteme in einem AS, die Pakete an Zielknoten außerhalb des autonomen Systems weiterleiten. Ein *Inter-AS-Routing-Protokoll* wird für Gateways benötigt, um Pakete von einem autonomen System an ein anderes weiterzuleiten.

Intra-AS-Routing
Gateway

Inter-AS-Routing

Dieser Ansatz begrenzt die Komplexität der Pfadberechnung, da innerhalb eines autonomen Systems nur die Größe des eigenen homogenen Netzwerks bewältigt werden muss. Zwischen autonomen Systemen müssen nur die Gateways mit der Anzahl verbundener anderer autonomer Systeme fertig werden. Die Betreiber von autonomen Systemen sind frei in der Wahl ihres Intra-AS-Routing-Protokolls, solange die Inter-AS-Routing-Protokolle auf den Gateways mit den Gateways der verbundenen anderen autonomen Systeme kompatibel sind.

Ein Gateway-Router muss daher sowohl das Intra-AS-Routing-Protokoll ausführen, damit es im autonomen System erreichbar ist, als auch das Inter-AS-Routing-Protokoll zur Kommunikation mit den Gateways der verbundenen anderen autonomen Systeme. Wenn ein Paket an eine Adresse außerhalb des eigenen autonomen Systems gesendet werden soll, dann muss das Paket lediglich innerhalb des autonomen Systems an ein passendes Gateway geschickt werden. Dieses Gateway wendet dann sein Inter-AS-Routing-Protokoll an und schickt das Paket zu einem anderen Gateway. Dieses schickt das Paket dann mit seinem Intra-AS-Routing-Protokoll im eigenen autonomen System weiter. Es ist wichtig anzumerken, dass ein Paket durchaus durch mehrere autonome Systeme geschleust werden kann, bevor es das autonome System erreicht, in dem sein Zielknoten liegt.

Wegen der so entstehenden Hierarchie von Routing (Intra-AS-Routing als Grundlage und darüber die Schicht des Inter-AS-Routing) nennt man diesen Ansatz auch *hierarchisches Routing*.

4.3 Die Vermittlungsschicht im Internet

In diesem Abschnitt betrachten wir die aktuelle Vermittlungsschicht des Internets, die *IP-Schicht*. Sie bietet einen verbindungslosen Datagramm-Dienst. Nach Übergabe eines Segments von der Transportschicht an die Vermittlungsschicht des Senders wird es in ein oder mehrere IP-Datagramme verpackt und mit den Adressen des Senders und Empfängers versehen. Jedes Datagramm wird an den ersten Router auf dem Pfad zum Empfänger geschickt.

Die Vermittlungsschicht im Internet besteht aus drei Komponenten, die wir in den folgenden Abschnitten genauer behandeln werden:

1. Das Internetprotokoll der derzeit noch oft benutzten Version *IPv4* [2] definiert die Adressierung in der Vermittlungsschicht, die Felder eines Datagrammes und die Aktionen, die durch Router und Endsysteme auf einem Datagramm in Abhängigkeit von den Inhalten seiner Felder ausgeführt werden.
2. Die *Pfadbestimmungskomponente* bestimmt den Pfad, den ein Datagramm auf dem Weg von dem Sender zum Empfänger durch das Internet nimmt.
3. Das Protokoll *ICMP* (*Internet Control Message Protocol* [14]) ist die Komponente des Internets, die es erlaubt, Fehler in Datagrammen anzuzeigen und Informationen über die Vermittlungsschicht abzufragen.

Begrenzung der Komplexität der Pfadberechnung

Rolle der Gateways

hierarchisches Routing

IP-Schicht

IPv4

Pfadbestimmungskomponente

ICMP

Übungsaufgabe 4.5 Wie viele Hosts kann es in einem Netzwerk der Klasse A höchstens geben?

<https://e.feu.de/1801-klasse-a-hosts>



Netzwerkpräfix

Host-Teil

Netzwerkmaske

Abbildung 4.10 zeigt ein Beispiel für die Interface-Adressen in drei miteinander verbundenen einfachen IP-Netzwerken, wobei das *Netzwerkpräfix* (*network prefix*) der Interfaces jeweils die ersten drei Byte sind. Das letzte Byte ist der *Host-Teil*.⁴ Das einfache IP-Netzwerk links in der Abbildung 4.10 hat die Adresse 220.1.1.0/24, wobei 220.1.1.0 die Basisadresse des Netzwerks ist. Die Notation /24 drückt aus, dass die ersten 24 Bit der Adresse die Netzwerkadresse definieren. /24 heißt auch manchmal *Netzwerkmaske* (*network mask, subnet mask*), die auch in der Form 255.255.255.0 dargestellt wird. Jedes Interface, das zu einem Netzwerk hinzugefügt werden soll, muss diesem Adressierungsschema folgen. Ein neues Interface im Netzwerk 220.1.1.0/24 muss also eine IP-Adresse der Form 220.1.1.x mit $x \in [0, 255]$ haben.

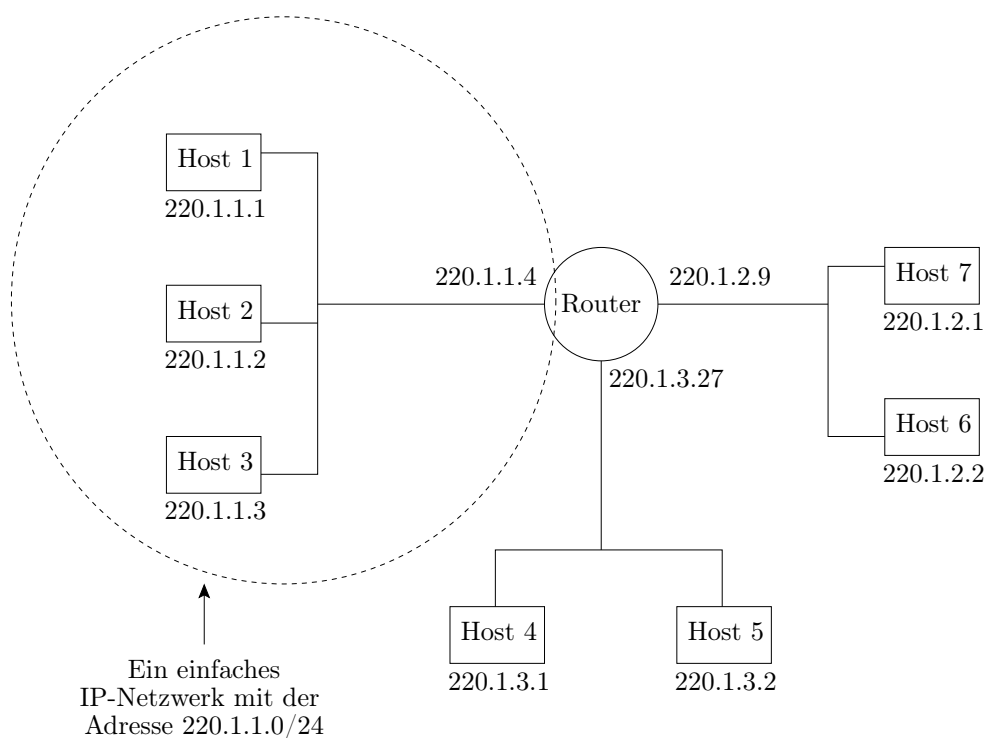


Abbildung 4.10: Interface-Adressen für ein Netzwerk aus drei einfachen Netzwerken.

Die Adressaufteilung in Abbildung 4.9 erwies sich aber als zu unflexibel, da in vielen Netzwerken große Mengen von Adressen ungenutzt blieben. Deshalb ist es mit dem *CIDR-Standard* (*Classless Interdomain Routing Standard*) der IETF [5] nun möglich, die 32 Bit der Adresse beliebig aufzuteilen: Die

CIDR-Standard

⁴Er sollte eigentlich besser Interface-Teil genannt werden.

DHCP-Protokoll

Notation $a.b.c.d/x$ meint, dass die ersten x der 32 Bit der Adresse das Netzwerkpräfix beschreiben. Einem Netzwerk mit z. B. weniger als 2000 aber mehr als 256 Hosts wären nach der alten Schema ein Netzwerkpräfix der Klasse B mit $2^{16} = 65536$ Host-Adressen zugeteilt worden; nun kann es eine Adresse $a.b.c.d/21$ bekommen, verwendet also nur die ersten 21 Bit als Netzwerkpräfix und hat $2^{11} = 2048$ interne Adressen zur Verfügung. Damit bleiben gegenüber früher etwa 63.000 IP-Adressen frei, die anderen Organisationen zugeteilt werden können.

Ein Host kann auf mehreren Wegen zu einer IP-Adresse beziehungsweise zum Host-Teil hinter dem ansonsten festliegenden Netzwerkpräfix kommen. Eine Möglichkeit ist die manuelle Konfiguration durch den Systemadministrator, der die IP-Adresse üblicherweise in eine Datei schreibt. Eine andere oft genutzte Möglichkeit ist die Nutzung des *DHCP-Protokolls* (*Dynamic Host Configuration Protocol* [4]). In einem Netzwerk kann ein Host eine Anfrage an einen DHCP-Server stellen, der dann dem Host eine freie IP-Adresse dynamisch zuordnet.

An eine Netzwerkadresse, also das Netzwerkpräfix, zu kommen, ist schon erheblich schwieriger. Ein Netzwerkadministrator bekommt eine Netzwerkadresse in der Regel von dem Internet-Provider der Organisation. Jeder Internet-Provider hat üblicherweise einen größeren Adressraum für seine Kunden reserviert. Diesen Adressraum kann der Provider unter seinen Kunden aufteilen. Internet-Provider müssen ihre Adressräume wiederum bei der *ICANN* (*Internet Corporation for Assigned Names and Numbers* [1]) bzw. bei den von ihr lizenzierten regionalen Internet Registries beantragen und dafür natürlich auch bezahlen.

ICANN



Übungsaufgabe 4.6 Eine Universität hat weniger als $2^5 = 32$ aber mehr als $2^4 = 16$ Fakultäten und jede besitzt mehr als $2^{10} = 1024$ aber weniger als $2^{11} - 2 = 2046$ Rechner.

Das Netzwerk der Universität hat einen Hauptrouter, der mit einem ISP oder einem regionalen Netzwerk verbunden ist, siehe Abbildung 4.11. Der Universität sind $2^{16} = 65536$ IP-Nummern der Klasse B mit der Adresse 135.50.0.0/16 zugeteilt worden.

Wie können diese IP-Adressen 135.50.0.0/16 auf die Fakultäten so verteilt werden, dass jede ein einfaches Teilnetzwerk innerhalb des Uni-Netzwerks mit einer sogenannten Teilnetzwerkmaske bildet? Geben Sie für die 8 Fakultäten aus Abbildung 4.11 die erste zugewiesene IP-Adresse sowie die Teilnetzwerkmaske in der Notation $a.b.c.d/x$ an.

<https://e.feu.de/1801-teilnetzwerke>



4.3.1.2 Routing von IP-Datagrammen

Jedes IP-Datagramm besitzt ein Feld für die IP-Adresse des Senders und ein Feld für die IP-Adresse des Empfängers. Der Sender trägt seine eigene IP-Adresse in das Sender-Adressfeld und die Empfänger-IP-Adresse in das

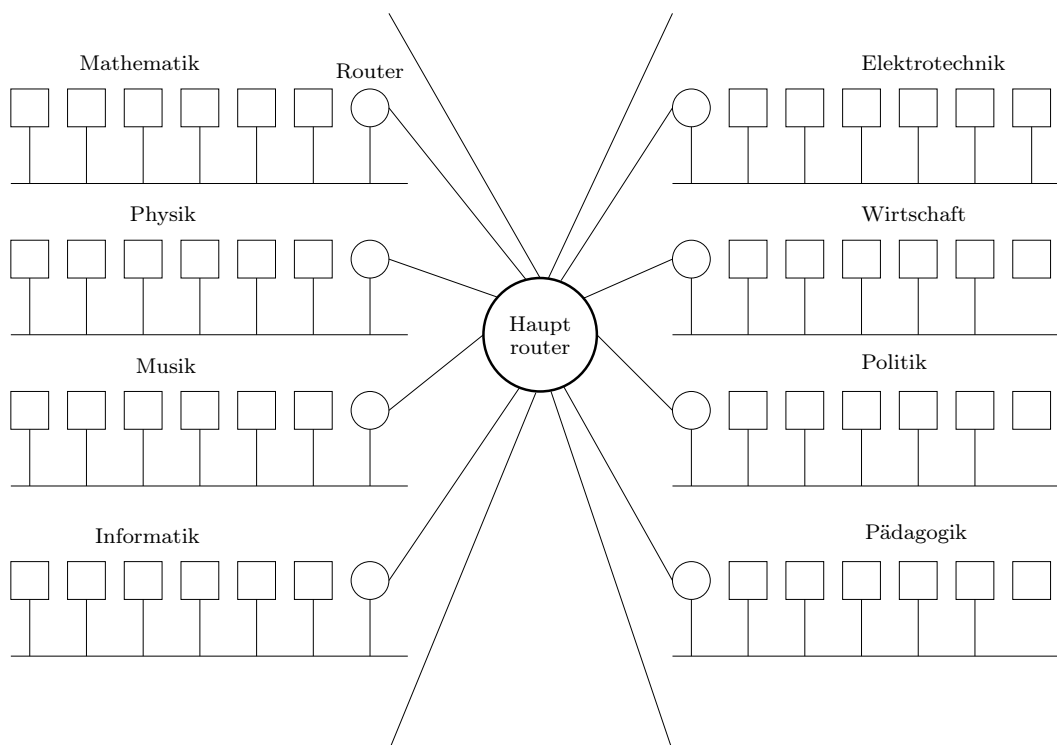


Abbildung 4.11: Das Netzwerk der Universität.

Empfänger-Adressfeld ein. In das Datenfeld kommt üblicherweise ein TCP- oder UDP-Segment. Das Längenfeld gibt an, wie viele Bytes tatsächlich genutzt sind. Die weiteren Felder und ihre Bedeutung können Sie bei Bedarf in der Literatur [9, 16] nachlesen.

Wir wollen den Transport eines IP-Datagramms durch die Vermittlungsschicht anhand eines Beispiels diskutieren, siehe Abbildung 4.12.

Wenn Host 1 ein IP-Datagramm an Host 3 schickt, der in demselben einfachen IP-Netzwerk liegt, dann passiert das folgende: Zuerst sucht das IP-Protokoll in der Routing-Tabelle von Host 1 nach einem Eintrag, dessen Netzwerkadresse zu der IP-Adresse von Host 3 passt. Das heißt, die Adresse 220.1.1.3 von Host 3 wird der Reihe nach mit den Einträgen in der Routing-Tabelle verglichen, und zwar nur in den führenden Bits der Netzwerkadresse, bis eine Übereinstimmung gefunden wird. In diesem Beispiel passt schon der erste Eintrag 220.1.1.0/24. Jetzt weiß Host 1, dass Host 3 zu demselben Netzwerk gehört. Die Distanzen in der Routing-Tabelle geben zusätzlich an, wie viele Netzwerke einschließlich des Zielnetzwerks die Pakete vom Sender bis zum Ziel überqueren müssen.⁵ Man spricht auch von der Anzahl der *Hops*. Der Eintrag für das Netzwerk 220.1.1.0/24 zeigt, dass die Distanz zu Host 3 genau 1 ist. Host 1 weiß jedenfalls, dass er Host 3 direkt über seine ausgehende Schnittstelle erreichen kann, ohne weitere Router zu Hilfe nehmen zu müssen. Host 1 gibt das IP-Datagramm damit an die Sicherungsschicht weiter, die es

Versand im
selben einfachen
IP-Netzwerk

Hops

⁵Vergleichen Sie diese Angaben mit den Ausführungen in Abschnitt 4.2.2, insbesondere Abbildung 4.3.

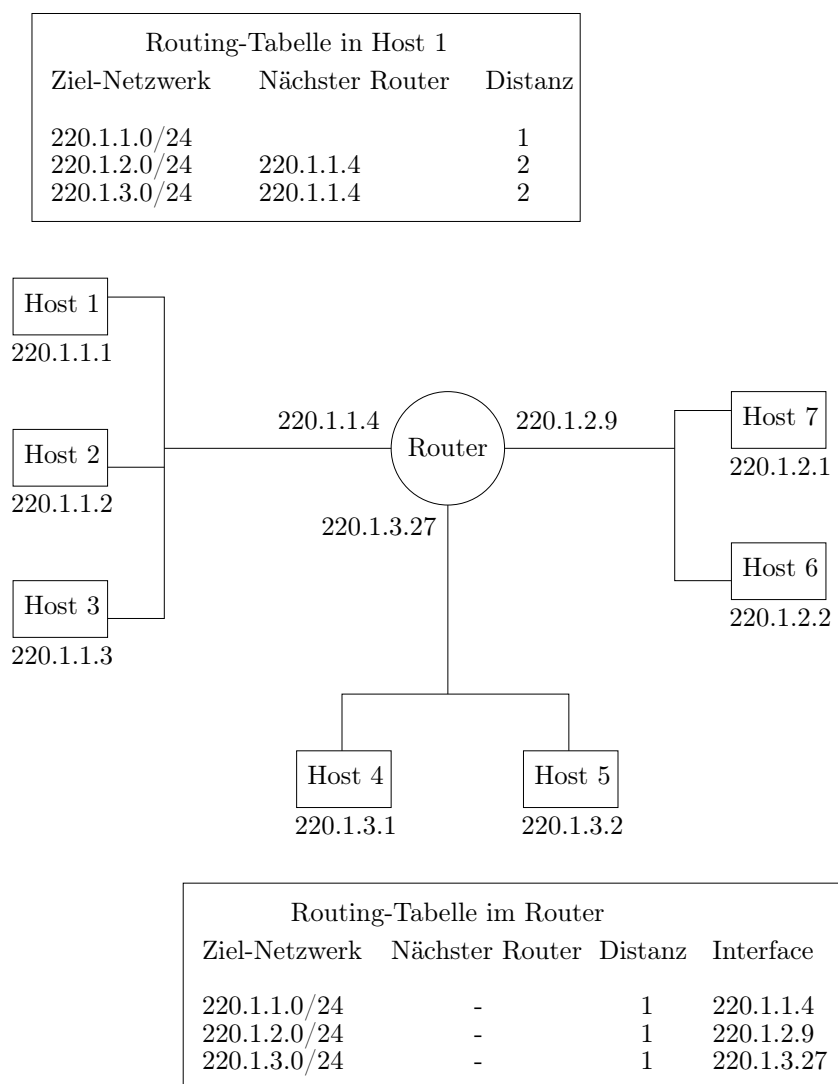


Abbildung 4.12: Routing-Tabellen in Host 1 und Router.

dann direkt zu Host 3 transportiert.

Wenn Host 1 ein IP-Datagramm an Host 5 sendet, der zu einem anderen Netzwerk gehört, dann wird der passende Eintrag 220.1.3.0/24 an der dritten Stelle der Routing-Tabelle gefunden. Jetzt sieht Host 1, dass er das IP-Datagramm an die IP-Adresse 220.1.1.4 der Schnittstelle des Routers schicken muss. Der Eintrag 220.1.3.0/24 zeigt noch, dass die Distanz zum Zielknoten 2 Hops beträgt, also liegt der Zielknoten in einem anderen Netzwerk, das nur über einen dazwischenliegenden Router erreicht werden kann. Die Sicherungsschicht überträgt das in einem Sicherungsschichtrahmen verpackte IP-Datagramm, ohne dass die Empfänger-IP-Adresse im IP-Datagramm verändert wird! Im Router angekommen, konsultiert der Router seine eigene Routing-Tabelle und verschickt das IP-Datagramm über seine Sicherungsschicht an den Empfänger. Der Eintrag in der Routing-Tabelle im Router hat hier durch die eingetragene Distanz 1 angezeigt, dass der Empfängerknoten direkt über das ausgehende Interface mit der angegebenen IP-Adresse 220.1.3.27 erreichbar ist.

Würde der Empfängerknoten in einem weiter entfernten Netzwerk liegen,

dann würde in der Routing-Tabelle des Routers ein Eintrag mit einer größeren Distanzzahl existieren und das Interface zum nächsten Router auf dem Pfad spezifizieren. Unser Router würde dann das IP-Datagramm an den nächsten Router weiterreichen usw. Aus dieser Diskussion wird deutlich, dass die Routing-Tabellen eine zentrale Rolle im Internet spielen. Es ist die Aufgabe der Routing-Algorithmen, möglichst gute Pfade in den Routing-Tabellen abzulegen, siehe auch Abschnitt 4.2.1 und 4.2.2.

4.3.1.3 Fragmentierung und Reassemblierung in IPv4

Sicherungsprotokolle, die die Vermittlungsschicht zum Transport der Datagramme über die physische Leitung benutzen müssen, unterscheiden sich unter anderem durch die maximale Paketgröße, und damit die maximale Anzahl an Datenbytes *MTU* (*maximum transfer unit*). Im Ethernet werden z. B. Pakete mit maximal 1500 Byte an Dateninhalt unterstützt. Dagegen unterstützen viele Verbindungen in Weitverkehrsnetzen Pakete mit höchstens 576 Byte Dateninhalt. Wenn ein IP-Datagramm über einen Pfad mit Verbindungen geleitet wird, die unterschiedliche MTUs unterstützen, dann tritt ein Problem auf: Was passiert mit einem Paket in der Sicherungsschicht, welches ein IP-Datagramm der Vermittlungsschicht enthält, wenn es plötzlich über eine Verbindung übertragen werden soll, die eine kleinere MTU hat?

maximale
Paketgröße MTU

Die Lösung dieses Problems liegt in der *Fragmentierung* und *Reassemblierung* von IP-Datagrammen: wann immer ein IP-Datagramm zu groß ist, wird es auf dem Router in mehrere kleinere IP-Datagramme zerlegt. Diese Fragmente können dann über die Verbindung mit der kleineren MTU übertragen werden. Bevor das ursprüngliche IP-Datagramm des Senderknotens an den Empfängerknoten übergeben werden kann, muss es aus den Fragment-IP-Datagrammen wieder zusammengesetzt werden. Dieses Zusammensetzen des ursprünglichen IP-Datagramms aus den Fragmenten passiert in der Vermittlungsschicht in den Endsystemen des Internets – *nicht* in den Routern. Damit dies funktioniert, muss ein Endsystem anhand der empfangenen IP-Datagramme erkennen, ob die empfangenen IP-Datagramme Fragmente eines größeren IP-Datagramms waren. Dazu dienen in IPv4 bestimmte Felder im Header der Datagramme: *Identification* (ID), *Flag* und *Fragmentation Offset*. Bei Erzeugung eines IP-Datagramms versieht der Sender dieses mit einer neuen ID durch Inkrementieren der ID des letzten gesendeten Datagramms. Wenn das IP-Protokoll eines Router ein IP-Datagramm fragmentiert, dann behält jedes Fragment *dieselbe* ID des Datagramms. Alle Fragmente haben das Flag-Feld auf 1 gesetzt, bis auf das letzte Fragment. Hieran erkennt das Endsystem, dass das letzte Fragment eines größeren IP-Datagramme empfangen wurde. Anhand des Fragmentation Offset kann das Endsystem die Fragmente in die richtige Reihenfolge bringen und Lücken erkennen. Das Feld *Fragmentation Offset* gibt an, an welche Stelle im aktuellen Datagramm das Fragment gehört. Falls ein oder mehrere Fragmente verloren gegangen sind, dann wird das gesamte ursprüngliche IP-Datagramm verworfen. Im Falle von TCP muss das Transportprotokoll dann eine Wiederholung veranlassen.

Verlangsamung
der Verarbei-
tungsprozesse

Die Fragmentierung verlangsamt die Verarbeitungsprozesse im Router, so wie die Reassemblierung die Verarbeitung auf dem Endsystem verlangsamt. Deshalb möchte man Fragmentierung vermeiden. Sofern man die Größe eines IP-Datagramms auf den kleinsten gemeinsamen Nenner verringert, kann man die Fragmentierung ganz verhindern. Im Internet sollen die eingesetzten Sicherungsschichtprotokolle MTUs von mindestens 576 Byte unterstützen. Deshalb kann eine Beschränkung des Dateninhalts auf 536 Byte pro IP-Datagramm sinnvoll sein. Zusammen mit dem 20 Byte IP-Header und 20 Byte für den Header des TCP-Segments liegt man dann unter der magischen Grenze von 576 Byte.

4.3.2 Pfadbestimmung im Internet

In diesem Abschnitt betrachten wir die *Routing-Protokolle des Internets*. Das Internet besteht aus einem Verbund von Netzwerken, die von regionalen Organisationen betrieben werden. In Abschnitt 4.2.4 wurde definiert, dass eine Menge von Routern, die von derselben Organisation betrieben werden und die dasselbe Routing-Protokoll benutzen, als autonomes System (AS) bezeichnet wird. Jedes autonome System besteht selbst wiederum aus einer Menge von einfachen IP-Netzwerken. Wir unterscheiden im Internet zwei Fälle des Routing:

1. In einem autonomen System: Intra-AS-Routing
2. Zwischen autonomen Systemen: Inter-AS-Routing

4.3.2.1 Intra-AS-Routing

Ein Routing-Protokoll für Intra-AS-Routing konfiguriert die Routing-Tabellen in allen Routern eines autonomen Systems. In der Praxis kommen im Internet vor allem drei Protokolle zum Einsatz: RIP (Routing Information Protocol [6, 11]), OSPF (Open Shortest Path First [12]) und EIGRP (Enhanced Interior Gateway Routing Protocol, ein proprietäres Protokoll von Cisco). Wir behandeln hier nur RIP.

RIP

RIP ist eines der frühesten Intra-AS-Routing-Protokolle, das heute noch viel genutzt wird. Es nutzt einen Distanzvektor-Algorithmus ähnlich zu dem in Abschnitt 4.2.2 beschriebenen. Die RIP Version 1 [6] verwendet als Kosten die Anzahl der Hops, die bei der Übertragung von Datenpaketen auf dem Weg zum Ziel durchlaufen werden müssen, siehe auch Abschnitt 4.3.1.2. Außerdem sind die maximalen Kosten eines Pfades auf den Wert 15 beschränkt. Deshalb können autonome Systeme mit RIP nur einen maximalen Durchmesser von 15 Routern haben. Nachbarknoten tauschen in RIP etwa alle 30 Sekunden Routing-Informationen mit sogenannten *RIP Response Messages* aus, indem sie sich ihre Routing-Tabelleneinträge für höchstens 25 Zielnetzwerke im autonomen System verschicken.

Betrachten wir nun den Ausschnitt eines autonomen Systems in Abbildung 4.13. Tabellen 4.2 und 4.4 zeigen die Entwicklung der Routing-Tabelle

von Router *B* aufgrund des Empfangs einer der RIP Response Message von Router *A*, die in Tabelle 4.3 dargestellt ist.

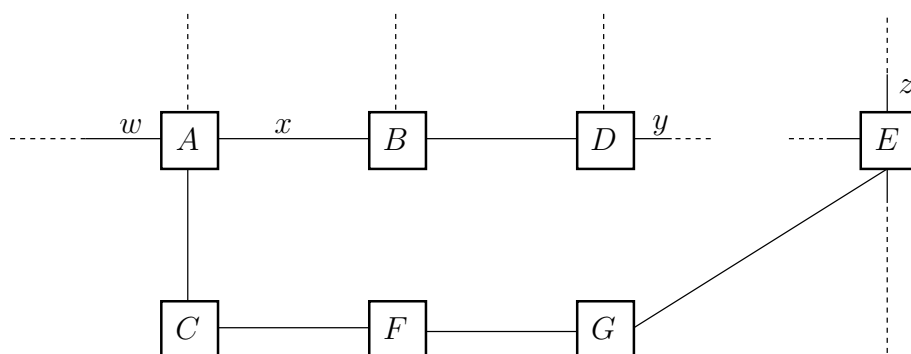


Abbildung 4.13: Ausschnitt aus einem autonomen System. Gestrichelte Linien zeigen Verbindungen zu weiteren Routern an, Kantenbezeichnungen *w*, *x*, *y*, *z* bezeichnen Netzwerke.

Zielnetzwerk	Nächster Router	Distanz zum Ziel
<i>w</i>	<i>A</i>	2
<i>y</i>	<i>D</i>	2
<i>z</i>	<i>D</i>	7
<i>x</i>	-	1
...

Tabelle 4.2: Ausschnitt aus der Routing-Tabelle von Router *B* vor dem Empfang der RIP Response Message von Router *A*.

Die Routing-Tabelle 4.2 ist folgendermaßen zu lesen: wenn Router *B* Pakete an das Netzwerk *w* verschicken will (siehe erste Zeile), dann versendet er diese über den Router *A*.

Nachdem Router *A* das Update (exakt seine Routing-Tabelle, siehe Tabelle 4.3) an *B* verschickt hat, erfährt *B*, dass das Netzwerk *z* nur 5 Schritte über Router *A* entfernt ist. Bisher war *z* aber über Router *D* mit der Distanz 7 erreichbar. Deshalb modifiziert *B* seinen Tabelleneintrag für *z* entsprechend (5 Schritte von *A* nach *z* plus 1 Schritt über die Verbindung *x* nach *A*), und sendet Pakete nach *z* nun über *A*. Wie kommt es in der Praxis zu einem Wechsel der Distanz zu einem Netzwerk? Entweder war der Algorithmus noch in der Konvergenzphase, oder es wurden neue Verbindungen oder Router zum autonomen System hinzugefügt.

Wenn RIP auf einem Router mehr als 180 Sekunden kein Update von einem Nachbarn erhält, betrachtet es den Nachbarn als unerreichbar. RIP modifiziert dann die lokale Routing-Tabelle und schickt Updates zu seinen Nachbarn. Ebenso können Router über RIP von Nachbarn deren Kosten zu einem Zielknoten abfragen.

Zielnetzwerk	Nächster Router	Distanz zum Ziel
z	C	5
w	-	1
x	-	1
...

Tabelle 4.3: Ausschnitt aus der Routing-Tabelle (bzw. der RIP Response Message) von Router A .

Zielnetzwerk	Nächster Router	Distanz zum Ziel
w	A	2
y	D	2
z	A	6
x	-	1
...

Tabelle 4.4: Ausschnitt aus der Routing-Tabelle von Router B nach dem Empfang der RIP Response Message von Router A .

RIP benutzt zur Kommunikation einzelne UDP-Datagramme über den Standard-Port 520. Es benutzt also ein Transportschichtprotokoll, UDP, um eine Funktion der Vermittlungsschicht zu realisieren, siehe auch Abschnitt 3.4.6. Der Grund hierfür liegt in der typischen Implementierung von RIP auf UNIX-Systemen als Anwendungsprozess (sogenannter `routed` daemon). Der `routed` Anwendungsprozess hat Zugang zu den Routing-Tabellen des Hosts, und kommuniziert über einen Socket mittels des Standard-Transportprotokolls UDP. RIP ist daher ein *Protokoll auf der Anwendungsschicht*.

4.3.2.2 Inter-AS-Routing

BGP4 (*Border Gateway Protocol Version 4* [15, 17]) ist der De-Facto-Standard für Inter-AS-Routing im Internet. Autonome Systeme werden im Internet als *administrative Domänen* (*Domains*) bezeichnet.

BGP basiert auf der Idee des Distanzvektor-Algorithmus, siehe Abschnitt 4.2.2. Allerdings ist BGP ein *Pfadvektor-Protokoll*, da BGP in einem Router keine Kosteninformationen (z. B. die Anzahl der Hops bis zum Zielknoten) ablegt, sondern Pfadinformationen, nämlich wie die Reihenfolge der autonomen Systeme auf einem Pfad vom Quell-AS zum Ziel-AS ist. BGP legt nicht fest, wie aus mehreren möglichen Pfaden zum Ziel-AS ein Pfad ausgewählt werden soll. Dies ist eine *Richtlinienentscheidung* (*Policy Decision*) des Domänenadministrators. Jede Domäne kann ihre eigenen Richtlinienentscheidungen treffen, ohne dass dies den anderen Domänen mitgeteilt würde.

BGP betrachtet das Internet als Graph aus autonomen Systemen, die durch Nummern bezeichnet werden. Zu einem bestimmten Zeitpunkt kann ein autonomes System X einen Pfad über mehrere AS zum autonomen System Y kennen – oder auch nicht. Angenommen, X hat den Pfad $X - Z_1 - Z_2 - Z_3 - Y$ zum autonomen System Y in der BGP-Routing-Tabelle. BGP kennt daher nicht nur

administrative
Domänen

BGP

den ersten Router auf dem Pfad, sondern alle Stationen. Wenn X ein Update zu seinen Nachbarn schickt, dann wird tatsächlich die gesamte Pfadinformation aus der Routing-Tabelle verschickt. Wenn ein Nachbar W von X nun dieses Update empfängt, dann kann er neue Einträge $W - X - Z_1 - Z_2 - Z_3 - Y$ zu seiner Tabelle hinzufügen – oder er kann diesen Update ignorieren, z. B. wenn er bereits einen besseren Pfad zu Y besitzt oder der neue Eintrag eine Routing-Schleife erzeugen würde.

BGP-Informationen werden durch das Netzwerk propagiert, indem die unmittelbaren Nachbarn im Graphen BGP-Nachrichten austauschen. Das BGP-Protokoll definiert vier Nachrichtentypen:

1. *Open-Nachricht*: BGP benutzt TCP als Transportprotokoll auf dem Port 179. Beim Verbindungsaufbau sendet BGP zuerst eine Open-Nachricht um sich beim Nachbarn zu authentifizieren⁶ und Informationen abzugleichen, z. B. Timer. Akzeptiert der Nachbar die Verbindung, sendet er eine KeepAlive-Nachricht zurück.
2. *Update-Nachricht*: Mit Update-Nachrichten werden Pfade an Nachbarn gemeldet oder zurückgezogen.
3. *KeepAlive-Nachricht*: Diese Nachricht dient zur Akzeptierung eines Verbindungsaufbaus durch eine Open-Nachricht, oder als Lebenszeichen, wenn der Sender keine neuen Informationen zu übertragen hat.
4. *Notification-Nachricht*: Diese Nachricht dient der Signalisierung eines Fehlers in der vorherigen Nachricht, oder zur Anzeige eines beabsichtigten Verbindungsabbaus.

In der Praxis hat ein autonomes System oft mehrere Gateway-Router, um Verbindungen zu anderen autonomen Systemen zu halten. Zwischen diesen Gateway-Routern innerhalb desselben autonomen Systems kann BGP verwendet werden, um Updates auszutauschen.

Übungsaufgabe 4.7 BGP benutzt den Distanzvektor-Algorithmus, um die Routing-Aufgabe zwischen autonomen Systemen zu lösen. Kann das *Count-to-Infinity-Problem* beim BGP entstehen?

<https://e.feu.de/1801-count2infinity>



4.3.3 Fehlerbehandlung und Abfragen im Internet

Im Internet benutzen Hosts, Router und Gateways das ICMP-Protokoll (Internet Control Message Protocol [14]) für den *Austausch von Vermittlungsschicht-Informationen*, wie z. B. Fehlermeldungen oder Informationen über benutzte Netzwerkpfade. ICMP benutzt zur Kommunikation IP-Datagramme, d. h.

⁶Sich ausweisen, also nachweisen, wer man ist.

ICMP-Nachrichten werden von ICMP direkt in IP-Datagramme verpackt. Konzeptionell ist ICMP daher ein Protokoll der über IP liegenden Schicht.⁷

ICMP-Nachrichten haben einen Typ und ein Feld namens *Code*, um die Bedeutung der Nachricht zu spezifizieren. Zusätzlich enthält eine ICMP-Nachricht die ersten 8 Byte des IP-Datagramms, welches zur Erzeugung der ICMP-Nachricht führte, siehe Tabelle 4.5, die einige ICMP-Nachrichten zeigt. Dies hilft dem Sender den vorliegenden Fehler zu erkennen.

ICMP-Typ	Code	Bedeutung
0	0	Echo-Antwort (auf eine Ping-Nachricht)
3	0	Zielnetzwerk nicht erreichbar
3	1	Zielhost nicht erreichbar
3	2	Ziel-Protokoll nicht erreichbar
3	3	Ziel-Port nicht erreichbar
3	6	Zielnetzwerk unbekannt
3	7	Zielhost unbekannt
4	0	Quelle reduzieren (zur Überlastkontrolle)
8	0	Echo-Anforderung (Ping-Nachricht)
9	0	Router-Bekanntmachung
10	0	Router-Entdeckung
11	0	Time-To-Live (TTL) abgelaufen
12	0	Fehlerhafter IP-Header

Tabelle 4.5: Einige ICMP-Nachrichtentypen.

ICMP wird oft für die *Kommunikation von Fehlermeldungen* benutzt: Falls ein Router feststellt, dass er keinen Weg zum Zielhost kennt, verwirft der Router das IP-Datagramm und erzeugt statt dessen eine ICMP-Fehlernachricht (z. B. Typ 3 mit Code 7 für Zielhost unbekannt). Wenn der Senderhost die ICMP-Fehlernachricht empfängt, reicht er diese als Fehlercode an das Transportprotokoll des Senders (z. B. TCP) zurück, das diesen Fehler dann an die Anwendung weiterreicht.

Beispiel: ping

Ein Beispiel für die Verwendung von ICMP ist das Programm **ping**. Es sendet eine ICMP-Nachricht vom Typ 8 mit Code 0 an den Zielhost. Der Zielhost empfängt die Echo-Anforderung und sendet eine Echo-Antwort (ICMP-Nachricht Typ 0 mit Code 0) zurück.

Beispiel:
tracert

Ein anderes Beispiel für den Einsatz von ICMP ist das Programm **tracert**. Es ermittelt den Pfad, den IP-Datagramme zu einem angegebenen Zielhost nehmen. Dazu sendet das Programm eine Reihe gewöhnlicher IP-Datagramme an den Zielhost. Dabei erhöht es bei jedem neuen IP-Datagramm das Feld TTL (Time-To-Live) um eins. Router, die das IP-Datagramm übertragen, verringern das TTL-Feld bei jeder Übertragung um 1. Wenn das TTL-Feld

⁷ICMP ist einerseits eine Anwendung, die direkt auf IP implementiert ist und kein gesondertes Transportprotokoll benutzt. Es gilt aber andererseits als Bestandteil der IP-Schicht, wie bereits eingangs erwähnt, siehe Abschnitt 4.3.

auf Null gesetzt wird, wird das IP-Datagramm verworfen.⁸ In diesem Fall aber sendet der Router eine Warnung (ICMP-Nachricht mit Typ 11 und Code 0) an den Senderhost, die auch die IP-Adresse des Routers und den Zeitstempel der Paketerzeugung enthält. Mit diesen Informationen kann ICMP dann den Pfad und die Umlaufzeit der Pakete auf dem Pfad berechnen und anzeigen.

4.4 Neuere Entwicklungen für die Vermittlungsschicht

Seit den frühen neunziger Jahren entwickelt die Internet Engineering Task Force (IETF) das IPv6-Protokoll [3, 7, 8] als Reaktion auf verschiedene Probleme mit IPv4:

- Aufgrund des rapiden Wachstums des Internets war absehbar, dass der Adressraum von IPv4 in relativ naher Zukunft erschöpft sein würde. Deshalb werden in IPv6 128 Bit lange Adressen (statt 32 Bit in IPv4) verwendet.
- Zusätzlich zu Unicast- und Multicast-Adressen werden sogenannte *Anycast-Adressen* eingeführt, mit denen ein Datagramm an irgendeinen Host aus einer Anycast-Gruppe weitergeleitet werden kann. Dies ist immer dann nützlich, wenn mehrere Knoten im Netzwerk denselben Dienst anbieten, und es der Anwendung egal ist, welcher Knoten den Dienst erbringt. Dies gilt z. B. für eine Mirror-Site eines Web-Servers oder die Beantwortung einer DNS-Anfrage durch einen DNS-Server. In diesem Fall können wir die Auswahl des Knotens aus der Anycast-Gruppe getrost der Vermittlungsschicht überlassen, die z. B. den am billigsten erreichbaren Knoten wählen könnte.
- Der 40 Byte lange IPv6 Header wurde so optimiert, dass er effizienter verarbeitet werden kann. So wurde zum Beispiel das Feld Prüfsumme entfernt, da die Transportprotokolle und viele Sicherheitsprotokolle selbst Fehlererkennung und Fehlerkorrektur durchführen. Dies führt dazu, dass in IPv6 das Berechnen und Testen der Prüfsumme in jedem Router entfallen kann. Weiterhin wurde auch das Fragmentation/Reassembly-Feld von IPv4 aufgegeben: ein Router, der ein zu großes Datagramm empfängt, verwirft dies einfach und sendet eine ICMP-Fehlermeldung an den Sender zurück. Dieser muss dann die Nachricht in kleinere Datagramme verpackt noch einmal schicken. Dies spart in den Routern den Aufwand für die Fragmentierung von Datagrammen ein.

längere Adressen

Anycast-Adressen

Optimierung des Headers

⁸Dieses Detail der IP-Vermittlung haben wir bisher unterschlagen: Durch das Herunterzählen des TTL-Feldes durch die Router werden IP-Datagramme, die ungewöhnlich lange unterwegs sind – weil sie z. B. in einer Routing-Schleife gefangen waren – erkannt. Diese überalterten Pakete werden bei Erreichen des Wertes Null automatisch eliminiert.

Der Übergang von IPv4 zu IPv6 kann auf verschiedene Weise erfolgen:

Dual-Stack
Ansatz

- An einem Stichtag könnte der Übergang aller Hosts und Router verpflichtend sein. Da das Internet aus Millionen von Geräten besteht, erscheint dieser Ansatz nicht sehr realistisch.
- *Dual-Stack Ansatz*: Jeder IPv6 Knoten könnte parallel eine IPv4-Implementierung und eine IPv6-Implementierung enthalten. Solche Knoten müssten mit Hilfe von DNS feststellen, welche IP-Version ein Empfängerknoten unterstützt und dann Pakete im entsprechenden Format senden. DNS kann einfach eine IPv4-Adresse für den Empfänger zurückgeben, wenn entweder der Sender oder der Empfänger nur IPv4 unterstützt. Dieser Ansatz führt allerdings dazu, dass Datagramme zwischen IPv6-Knoten, wenn sie über einen Pfad mit mindestens einem IPv4-Knoten transportiert werden, die Vorteile von IPv6 nicht ausnutzen können.

Tunneling

- *Tunneling Ansatz*: Hierbei können die IPv6-Knoten zwischen sich Tunnel aufbauen, indem sie den IPv6-Verkehr bei Übertragung über IPv4-Knoten insgesamt in IPv4 Datagramme einpacken. Hierbei kann der IPv6-Empfänger dann das IPv6-Datagramm wieder auspacken und normal weiterverarbeiten.

Es bleibt zum Schluss anzumerken, dass der Austausch eines Protokolls der Vermittlungsschicht sehr viel schwieriger ist, als ein neues Protokoll auf der Anwendungsschicht, wie z. B. HTTP, Chat oder Audio/Video Streaming, einzuführen. Dies liegt daran, dass die Vermittlungsschicht das Internet zusammenhält.

4.5 Aufgaben der Sicherungsschicht

Wie wir in den vorhergehenden Abschnitten gesehen haben, bietet die Vermittlungsschicht einen Kommunikationsdienst, der Datagramme von einem Host über eine Route zu einem anderen überträgt. Die Route besteht aus einer Reihe von direkten Verbindungsleitungen zwischen Netzknoten, das können Hosts oder Router sein. In der *Sicherungsschicht* (*data link layer*) geht es nun darum, wie Datagramme der Vermittlungsschicht von *einem* Knoten zu einem anderen *direkt benachbarten* Knoten über eine einzelne Verbindungsleitung übertragen werden.

Aus Platzgründen können wir in diesem Kurs nur einen kurzen Überblick über die Sicherungsschicht geben. Im Kurs *Kommunikations- und Rechnernetze* wird die Sicherungsschicht detailliert behandelt.

Die heute übliche Form der Vernetzung von mehr als zwei Rechnern über eine physische Leitung sind Busstrukturen, und das gebräuchlichste Protokoll ist das Ethernet-Protokoll. Der Bus ist ein allen Knoten gemeinsam zugängliches Übertragungsmedium, das einen passiven Nachrichtentransport unterstützt. Der Sender sendet die Bits in Form von Signalen, und während ein Bit an einem Knoten vorbeifließt, strömt ein Teil der Signalenergie in den Adapter (Busankopplung). Terminatoren an beiden Busenden schließen den Bus ab und

Kommunikation
über einzelne
Verbindungs-
leitungen

absorbieren die Restenergie der Signale, siehe Abbildung 4.14. So hören alle angeschlossenen Knoten die von einem beliebigen Knoten gesendeten Signale mit. *Busnetze* heißen daher auch *Broadcast-Netze* oder *Carrier-Sense-Netze*. Broadcast-Netze haben einen einzigen Übertragungskanal, der von allen am Netz angeschlossenen Hosts gemeinsam genutzt wird. Pakete werden von einem Host versendet und von allen anderen empfangen. Zugangsprotokolle für Broadcast-Netze regeln den Zugriff von Knoten auf den Bus sowie das Verhalten im Falle von Kollisionen, wenn verschiedene Knoten gleichzeitig versuchen, den Bus zu nutzen. Die physische Ausprägung eines solchen Netzes kann verschiedene Formen annehmen, indem Busnetze durch sogenannte *Repeater* miteinander verbunden werden, siehe Abbildung 4.15. Ein Repeater ist ein reiner Signalverstärker, der ankommende Signale aus einem angeschlossenen Segment unverändert an alle anderen angeschlossenen Segmente weiterleitet. Das klassische Ethernet z. B. unterstützte bis zu vier Repeater, um die maximale Kabellänge von 500 m auf 2.500 m zu erweitern.

Busnetze
Broadcast-Netze

Repeater

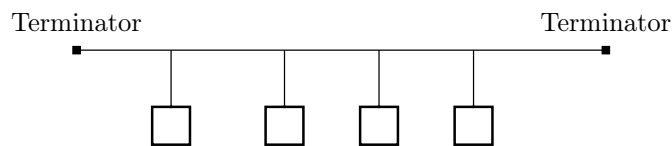


Abbildung 4.14: Einfaches Busnetz.

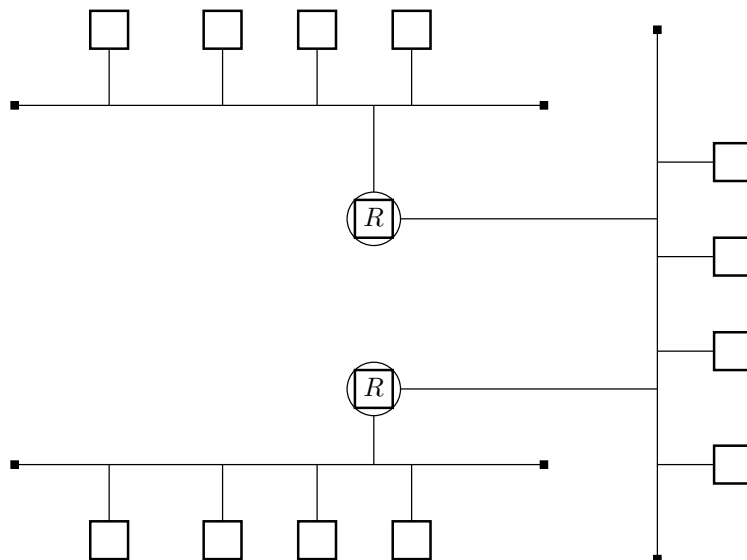


Abbildung 4.15: Durch Repeater (*R*) erweiterte Busstruktur eines Busnetzes.

Ringnetz

In den 80ern und Anfang der 90er Jahre wurden verstärkt *Ringnetze* genutzt, ein zweiter Typ von Broadcast-Netzen. Die Datenübertragung erfolgt hier über gerichtete Leitungen, die jeweils zwei benachbarte Knoten verbinden, siehe Abbildung 4.16 (a). Jeder Knoten besitzt eine Ringankopplung, die aktiv am Datentransport beteiligt ist. Sie entscheidet, ob eine Nachricht unverändert weiterzugeben (Broadcast), verändert weiterzuleiten oder vom Ring zu nehmen ist. Protokolle für Ringnetze sind Token-Ring (IEEE 802.5) und FDDI. Im Unterschied zu den Busankoppelungen in Busnetzen sind die Ringankoppelungen aktiv am Transport beteiligt. Dadurch fallen Verzögerungszeiten pro Knoten an. Ein weiteres Merkmal von Ringnetzen ist, dass bei Ausfall eines Knotens keine geregelte Kommunikation mehr möglich ist. Dieses Problem tritt in einem *sternförmigen Netz* nicht mehr auf, wo jeder Knoten direkt mit einem zentralen Gerät – dem Hub – verbunden wird, siehe Abbildung 4.16 (b). So können ausgefallene Knoten die Kommunikation nicht mehr behindern. Auch das Einfügen neuer Knoten in ein Netz wird vereinfacht.

sternförmiges
Netz

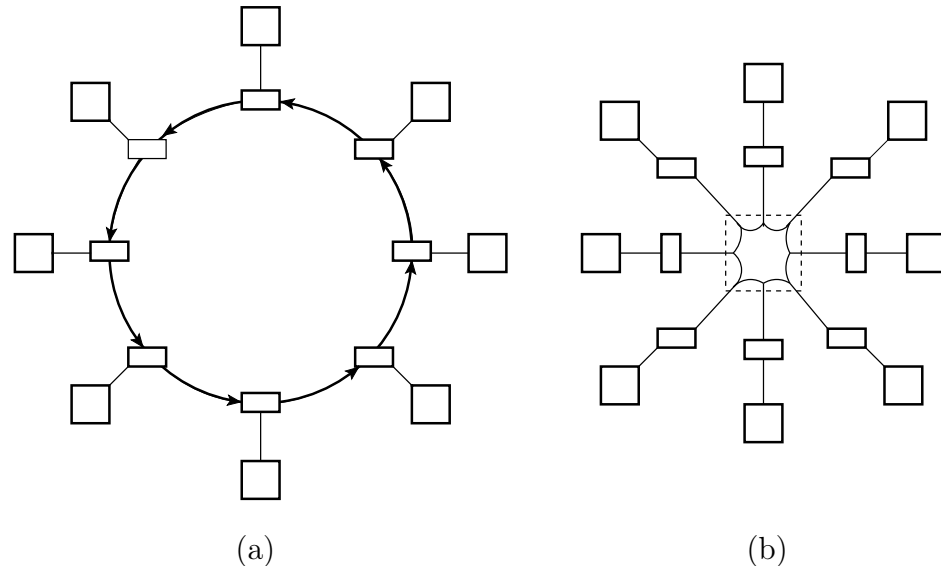


Abbildung 4.16: (a) Ringnetz, (b) sternförmiges Netz mit einem Hub in der Mitte.

Adapter

Protokolle der Sicherungsschicht werden also in der Regel für eine bestimmte Kommunikationsleitung implementiert. Sie sind in Adaptern, als *Netz-schnittstellenkarte* (*Network Interface Card, NIC*) realisiert, die heute meistens als Steckkarte (oder PCMCIA-Karte) angeboten wird. Ein *Adapter* verfügt über *zwei* Komponenten, das *Bus-Interface* zu dem Rechner, auch *Host-Bus-Schnittstelle* genannt und das *Leitungs-Interface* zu der Netzwerkleitung, auch *Leitungsschnittstelle* genannt, siehe Abbildung 4.17.⁹ Er verbindet also Host oder Router mit der physischen Netzwerkleitung.

⁹Die Benutzung des Begriffes Bus mag etwas verwirrend sein. Das Bus-Interface verbindet nicht die Rechner über eine Leitung, die eine Busstruktur haben kann, sondern bildet die Schnittstelle zu dem Gerät, in dem der Adapter steckt! Wie andere Ein-/Ausgabegeräte ist auch der Adapter über einen E/A-Bus im Rechner mit Prozessor und Hauptspeicher verbunden, vgl. Abschnitt 1.3.

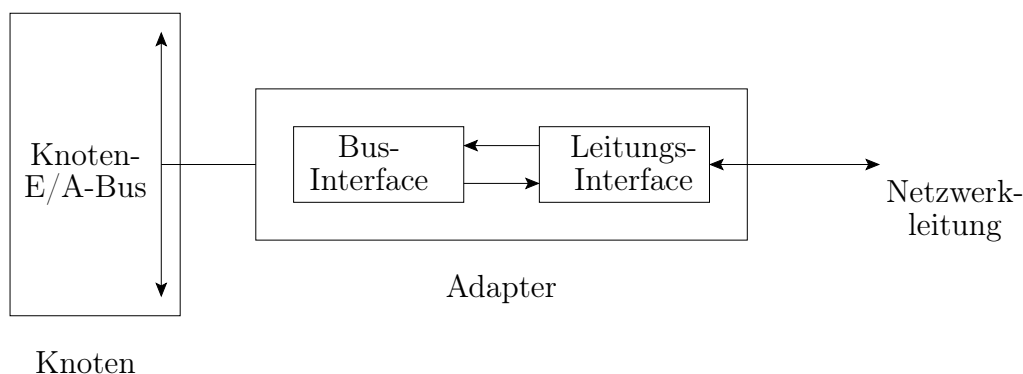


Abbildung 4.17: Aufbau eines Adapters.

Adapter werden auch als *halbautonome Einheiten* bezeichnet: Alle Entscheidungen bezüglich des Sicherungsschichtprotokolls sind in der Leitungsschnittstelle des Adapters, oft in Hardware, implementiert.¹⁰ Der Adapter erhält zu versendende Datagramme des Knotens über den E/A-Bus des Knotens und sein Bus-Interface, verpackt diese in seinem Leitungs-Interface in Rahmen der Sicherungsschicht und schickt den Rahmen mit Hilfe der unterliegenden Bitübertragungsschicht auf die Netzwerkleitung. Erhält der Adapter Daten von der Netzwerkleitung, spricht er den Knoten nur über die Bus-Schnittstelle an, wenn Datagramme an die Schicht 3 weitergegeben werden sollen. Der Adapter gehört aber zum gleichen physischen Gerät wie der Knoten und teilt sich mit ihm Stromversorgung und Busse und steht unter der Kontrolle des Knotens, den man deshalb auch *Elternknoten* des Adapters nennt.

Die *Dienste eines Sicherungsschichtprotokolls* umfassen folgende Aspekte: Zum Weiterleiten der Datagramme werden die Datagramme in *Rahmen* (*frames*) gekapselt. Diese Aufgabe heißt auch *Framing*. Die eigentliche Weiterleitung soll nach Möglichkeit zuverlässig sein. Hier kommen Techniken, die wir schon von der Transportschicht kennen, wie das Bestätigen und die Neuübertragung zum Einsatz. Der Dienst der *Flusskontrolle* soll, wie auf der Transportschicht, siehe Abschnitt 3.4.5.6, den sendenden Knoten daran hindern, den empfangenden Knoten zu überschwemmen. Zudem kann ein Sicherungsprotokoll eine *Vollduplex- oder Halbduplex-Dienst* anbieten: im Halbduplex-Betrieb kann ein Knoten nicht gleichzeitig senden und empfangen. Bei Vollduplex können beide Knoten an den Enden einer Verbindungsleitung gleichzeitig Pakete übertragen. Auf diese Dienstaspekte werden wir nicht im Detail eingehen. Wir stellen drei typische Dienstaspekte von Protokollen der Sicherungsschicht vor. Die Unterschiede der von Schicht-2-Protokollen angebotenen Dienste resultieren im wesentlichen aus den Unterschieden der Leitungstypen, über die die Protokolle operieren:

- Von ihren physikalischen Eigenschaften her können störungsanfällige Leitungen häufig übertragene Bits verfälschen. Also können *Fehlererkennung* oder sogar Fehlerkorrektur das Weiterleiten verfälschter Rahmen und

halbautonome Einheit

Elternknoten
Dienste eines
Sicherungsschichtprotokolls
Framing

Flusskontrolle

Vollduplex- oder
Halbduplex-
Dienst

Fehlererkennung

¹⁰Ethernet-Adapter sind zum Beispiel sehr preisgünstig, da die Leitungsschnittstelle im Chip-Set implementiert ist.

Mehrfachzugriffsproblem	<p>damit das unnötige Belasten von Leitungen vermeiden. Für die Sicherungsschicht wurden deshalb ausgefeilte Techniken zur Fehlererkennung entwickelt, die wir hier nicht weiter behandeln können.</p> <ul style="list-style-type: none"> • Liegt beispielsweise genau eine einzige Verbindung zwischen einem Sender und einem Empfänger vor, ist der Leitungszugang trivial. Teilen sich jedoch mehrere Knoten eine Broadcast-Leitung, tritt das <i>Mehrfachzugriffsproblem</i> auf und Kanalzugriffsprotokolle müssen den Zugang zu der Leitung regeln.
Adressierung	<ul style="list-style-type: none"> • Die gemeinsame Nutzung eines Broadcast-Kanals durch mehrere Knoten erfordert zudem die Einführung von Adressen auf der Sicherungsschicht. Diese müssen mit den Adressen der Vermittlungsschicht in Deckung gebracht werden. Das Address Resolution Protocol (ARP, RFC 826, [13]) bildet im Internet die Adressen der Vermittlungsschicht auf Adressen der Sicherungsschicht ab.

4.6 Aufgaben der Bitübertragungsschicht

Die Bitübertragungsschicht stellt physische Übertragungskanäle für die Übertragung beliebiger Bitfolgen zur Verfügung. Wir werden auf die Bitübertragungsschicht nicht im Detail eingehen und nur einige Aspekte beleuchten.

Auswirkung der Bitübertragung auf obere Schichten	<p>Die Qualität des Übertragungskanals, den die Bitübertragungsschicht auf einem physischen Medium realisiert, hat immer Konsequenzen für die Entwicklung von Protokollen. Wir haben in den vorhergehenden Abschnitten entsprechende Konsequenzen gesehen: Die <i>Fehlerrate</i> des physischen Mediums begründet das ganze Spektrum der Fehlerbehandlung in Protokollen. Kupferkabel ist zum Beispiel rauschbehaftet und störungsanfällig während Glasfaser relativ sicher ist. Auf Kupferkabel verwendet man elektrische Signale zur Übertragung, auf Glasfaser optische Signale. Protokolle werden für diese unterschiedlichen Medien ausgelegt. Die maximal mögliche <i>Übertragungsgeschwindigkeit</i> des Mediums (gemessen in bit/s) bestimmt unter anderem die Leistungsfähigkeit der Datenübertragung. Die <i>Übertragungsverzögerung</i> führt dazu, dass Kollisionen entstehen können, obwohl die Sender die Leitung vor dem Senden abhören. Auch auf die <i>Abschwächung</i> von Signalen in Abhängigkeit des konkreten Mediums wurde bereits in den vorhergehenden Abschnitten eingegangen. Hierzu haben wir als Geräte der Bitübertragungsschicht Repeater kennen gelernt.</p>
---	--

Darstellen, Erkennen und Lesen von Information	<p>Zur Bitübertragung muss die Bitübertragungsschicht die zu übertragende Information auf dem physischen Medium darstellen können und beim Empfang wieder erkennen und herauslesen können. Die Information muss auf Signale, die mit dem physischen Medium übertragen werden können, abgebildet werden. Prinzipiell hat man bei der Informationsübertragung auf physischen Medien folgende Ausgangssituation zu betrachten: Die Daten liegen entweder digital (Texte oder Bitmaps) oder analog (Audio oder Video) vor. Zur Datenübertragung kann man ebenfalls digitale oder analoge Signale verwenden.</p>
--	---

Möchte man digitale und analoge Daten über einen Übertragungskanal übertragen, so muss man entweder digitale Daten auf analoge Signale abbilden oder analoge Daten auf digitale Signale. Das *Trägerstrom-Übertragungsverfahren* basiert darauf, digitale Informationen auf analogen Signalen darzustellen. Hier werden drei Modellierungsarten unterschieden: Amplituden-, Frequenz- und Phasenmodulationsverfahren, siehe Abbildung 4.18. *Modems* (*Modulator/Demodulator*) wandeln ein digitales Signal in ein analoges Signal um und gewinnen aus dem analogen Signal die digitale Information. So wurden die Telefonnetze die ersten Datenübertragungskanäle für jedermann.

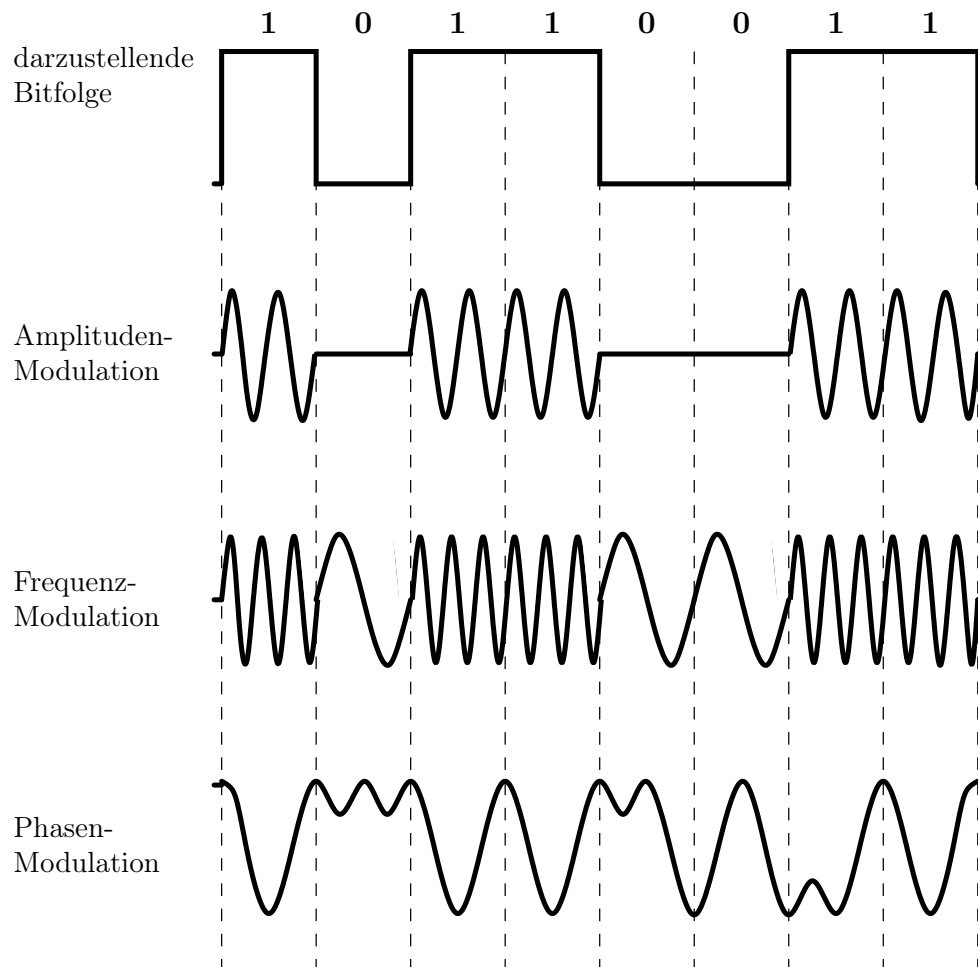


Abbildung 4.18: Modellierung von Nullen und Einsen auf analogen Signalen.

Während der Übertragung werden die Signale mehr oder weniger gedämpft oder gestört. Digitale Signale erlauben nicht nur die Verstärkung von Signalen, um Dämpfungen auszugleichen, sondern auch die Regeneration von Signalen, d.h. das Eliminieren von Störungen während der Übertragung. Dies erklärt den Erfolg der digitalen Übertragungstechnik. Analoge Daten werden dazu digital codiert. Dies ist auch die Grundlage der heutigen CD- und DVD-Technik für Audio- und Video-Daten. Ein bekanntes Verfahren ist das *PCM-Verfahren* (*Pulse Code Modulation*), siehe Abbildung 4.19. Datenübertragungs-

digitale Daten
und analoge
Signale

Modem

analoge Daten
und digitale
Signale

Codec

einrichtungen, die ein analoges Signal in ein digitales umsetzen beziehungsweise aus einem digitalen Signal ein analoges Signal wieder gewinnen, heißen *Codec* (*Codierer/Decodierer*).

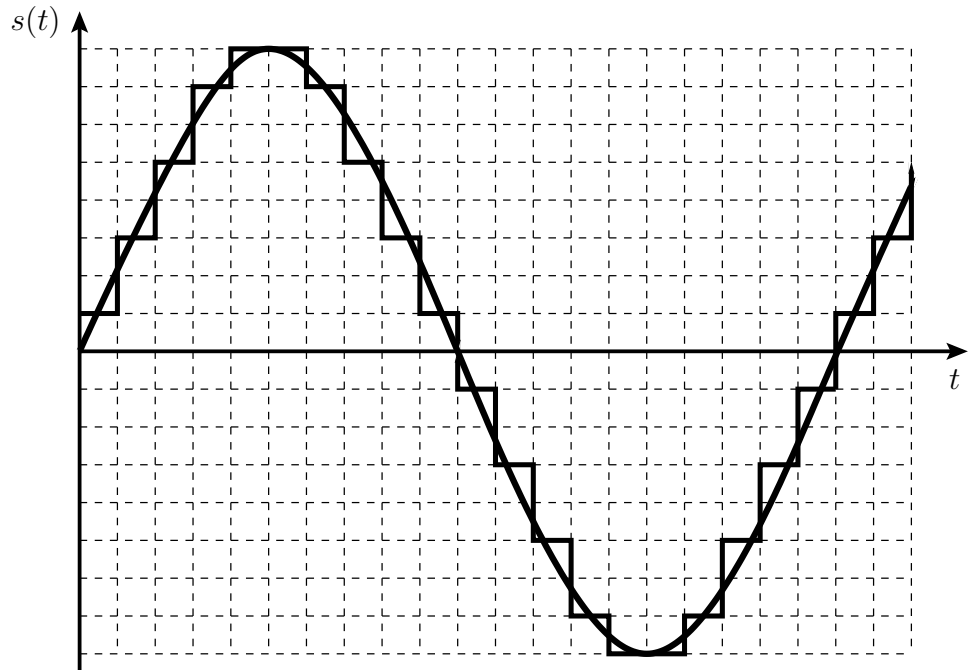


Abbildung 4.19: PCM-Verfahren: Die reellen Werte des analogen Signals werden gerundet und diese gerundeten Werte digital als Zahlen übertragen.

Synchronisation

Um die Signale zu erkennen, müssen sich Sender und Empfänger synchronisieren. Sender und Empfänger legen dazu die Schrittdauer und die Länge der Zeichen vorher fest. Beim *asynchronen Start-/Stopbetrieb* werden Beginn und Ende einer Teilübertragung dem Empfänger durch besondere Signalfolgen mitgeteilt. Im Falle der *synchronen Datenübertragung* wird ein Takt entweder zentral vom Netz geliefert oder aber der empfangene Rechner muss die Möglichkeit haben, sich von Zeit zu Zeit anhand des empfangenen Signals mit dem sendenden Rechner zu synchronisieren. Dazu wird der kontinuierliche Datenstrom in Blöcke fester Länge eingeteilt. Jeder Block beginnt mit einem Synchronisationszeichen, das ausschließlich zur Synchronisation beider Rechner dient. Ethernet führt ein solches Zeichen in Form der Präambel ein. Wegen der fehlenden Start/Stop-Schritte ist die Geschwindigkeit, mit der ein Zeichen im Mittel übertragen werden kann, bei synchroner Betriebsweise erheblich höher als bei asynchroner Betriebsweise.

4.7 Vertiefungen

In den Kurseinheiten 3 und 4 haben wir einen Überblick über den Aufbau und die Grundprinzipien von Rechnernetzen gewonnen.

Kurseinheit 3 begann mit einem Überblick über Rechnernetze und das Internet-Schichtenmodell. Es behandelte dann die obersten zwei Schichten, die Anwendungsschicht und die Transportschicht.

Kurseinheit 4 behandelte die tieferen Schichten des Schichtenmodells und Aspekte des Netzkerns. Die Prinzipien und Konzepte der Vermittlungsschicht bildeten den weitaus größten Teil dieser Kurseinheit, und haben ihr den Titel Vermittlung in Rechnernetzen gegeben. Um die Behandlung der Schichten des Protokollstapels abzuschließen, haben wir am Ende der Kurseinheit kurz die Sicherungsschicht und Bitübertragungsschicht besprochen.

Natürlich kann das Thema Rechnernetze nicht in zwei Kurseinheiten erschöpfend behandelt werden. Es ist unser Ziel, Ihnen in diesen beiden Kurseinheiten ein grundlegendes und praktisch orientiertes Verständnis des Aufbaus und der Funktionsweise von Rechnernetzen und der damit zusammenhängenden Probleme zu vermitteln. Außerdem wollen wir die Benutzung von Rechnernetzen zur Realisierung verteilter Anwendungen an einigen Beispielen illustrieren. Insbesondere haben wir in diesem Kurs eine ganze Reihe von Aspekten von Rechnernetzen nicht behandelt. Hierzu gehören z. B. die Multimedia-Vernetzung, Sicherheit in Computernetzwerken und Netzwerkmanagement. Auf der Basis der in diesem Kurs erworbenen Kenntnisse sollten Sie in der Lage sein, sich mit Hilfe der weiterführenden Literatur (z. B. [10, 16]) in diese Themen einzuarbeiten.

Wenn Sie dieses Thema im weiteren Verlauf Ihres Studiums vertiefen wollen, dann können Sie eine Reihe weiterführender Kurse belegen. Der Kurs *Betriebssysteme* betrachtet den Aufbau und die Funktionsweise moderner Betriebssysteme. Im Kurs *Verteilte Systeme* wird genauer dargestellt, wie verteilte Systeme auf der Basis moderner Betriebssysteme und Rechnernetze entworfen und realisiert werden können. Der Kurs *Kommunikations- und Rechnernetze* betrachtet mit mehr Detail, wie Netzwerke entworfen werden und wie sie funktionieren. Schließlich betrachtet der Kurs *Sicherheit im Internet*, mit welchen Mechanismen im Internet Sicherheit gegen Spione und Einbrecher gewährleistet werden kann.

--	--

Literatur

- [1] Internet Corporation for Assigned Names and Numbers, ICANN.
<http://www.icann.org>.
- [2] Internet protocol (IP-Version 4).
<http://www.rfc-editor.org/rfc/rfc791.txt>.
- [3] S. Bradner, A. Mankin. The recommendation for the IP next generation protocol. <http://www.rfc-editor.org/rfc/rfc1752.txt>.
- [4] R. Droms. Dynamic host configuration protocol (DHCP).
<http://www.rfc-editor.org/rfc/rfc2131.txt>.
- [5] V. Fuller, T. Li, J. Yu, K. Varadhan. Classless Inter-Domain Routing (CIDR): An Address Assignment and Aggregation Strategy.
<http://www.rfc-editor.org/rfc/rfc1519.txt>, 1993.
- [6] C. Hedrick. Routing Information Protocol (RIP).
<http://www.rfc-editor.org/rfc/rfc1058.txt>, 1988.
- [7] R. Hinden, S. Deering. IP Version 6 Addressing Architecture.
<http://www.rfc-editor.org/rfc/rfc2373.txt>.
- [8] R. M. Hinden, Sun Microsystems, Inc. IP Next Generation Overview. The Internet Engineering Task Force (IETF),
<https://tools.ietf.org/html/draft-hinden-ipng-overview-00>, October 1994.
- [9] J. F. Kurose, K. W. Ross. *Computernetze: Ein Top-Down-Ansatz*. Pearson Studium, 6. Auflage, 2014.
- [10] J. F. Kurose, K. W. Ross. *Computer Networking: A Top-Down Approach*. Pearson International Edition, seventh edition, 2017.
- [11] G. Malkin. RIP Version 2, Carrying Additional Information.
<http://www.rfc-editor.org/rfc/rfc1723.txt>, 1994.
- [12] J. Moy. OSPF Version 2. <http://www.rfc-editor.org/rfc/rfc2178.txt>.
- [13] D. C. Plummer. An Ethernet Address Resolution Protocol.
<http://www.rfc-editor.org/rfc/rfc826.txt>.
- [14] J. Postel. Internet Control Message Protocol (ICMP).
<http://www.rfc-editor.org/rfc/rfc792.txt>.

- [15] Y. Rekhter, T. Li. A Border Gateway Protocol 4 (BGP-4).
<http://www.rfc-editor.org/rfc/rfc1771.txt>.
- [16] A. S. Tanenbaum. *Computernetzwerke*. Pearson Studium, 5.,
aktualisierte Auflage, 2012.
- [17] P. Traina. Experience with the BGP-4 protocol.
<http://www.rfc-editor.org/rfc/rfc1773.txt>.