Prof. Dr.-Ing. Georg Carle
Dipl.-Ing. Stephan M. Günther, M. Sc.
Nadine Herold, M. Sc.
Dipl.-Inf. Stephan-A. Posselt
Johannes Naab, M. Sc.
Marcel von Maltitz, M. Sc.
Lehrstuhl für Netzarchitekturen und Netzdienste

Grundlagen Rechnernetze und Verteilte Systeme

Sommersemester 2014

— **Entwurf** —

Fakultät für Informatik
Technische Universität München

Vorwort

Das Skript zur Grundlagenvorlesung Rechnernetze und Verteilte Systeme befindet sich noch in der Entstehung. Wir bitten daher um Verständnis, dass es in diesem Semester noch in unvollständiger Form vorliegt.

Aufgrund des frühen Stadiums sind Fehler im Skript natürlich nicht auszuschließen. Sollten Sie einen Fehler vermuten oder auch allgemeine Anmerkungen und Verbesserungsvorschläge für das Skript haben, wenden Sie sich bitte an die Übungsleitung (mailto:grnvs@net.in.tum.de). Sollten Sie Widersprüche zwischen den Vorlesungsfolien und dem Skript feststellen, dann haben die Vorlesungsfolien Priorität.

Das Skript besteht im Wesentlichen aus dem Inhalt der Vorlesungsfolien. Die Folien wurden automatisch konvertiert und weisen leider Formatierungsfehler auf. Zum Lernen sollte der ursprüngliche Foliensatz verwendet werden, den man als separate PDFs auf der Webseite zur Vorlesung herunterladen kann:
http://www.net.in.tum.de/en/teaching/ss14/vorlesungen/

Inhaltsverzeichnis

1. Physikalische Schicht

1.1. Signale, Information und deren Bedeutung

1.1.0.1. Signale, Information und deren Bedeutung

Definition (Signale, Symbole):

Signale sind zeitabhängige und messbare physikalische Größen.
Definierten messbaren Signaländerungen lässt sich ein Symbol zuordnen.
Diese Symbole repräsentieren Information.

Beispiele für Signale:

- Licht (z.B. Übermittlung von Morsezeichen in der Schifffahrt)
- Spannung (z.B. Telegraphie)
- Schall (z.B. gesprochene Sprache; Musik)

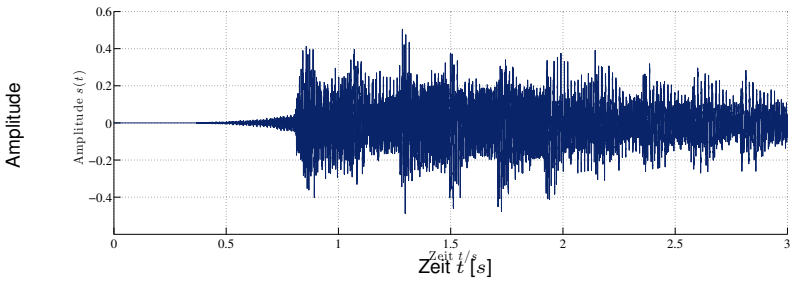


Abbildung 1.1.: Die ersten 3 s von „Sunrise Avenue – Hollywood Hills“

Definition (Informationstheorie nach Shannon):

Ansatz, den Begriff der Information statistisch zu erfassen.
Der Informationsgehalt eines Zeichens drückt aus, wieviel Information durch das Zeichen übertragen wird.

Der Informationsgehalt besitzt folgende Eigenschaften:

- Je seltener ein Zeichen auftritt, desto höher ist sein Informationsgehalt.
- Der Informationsgehalt einer Zeichenkette ist die Summe der Informationsgehalte der einzelnen Zeichen
- Der Informationsgehalt eines vorhersagbaren Zeichens ist 0

Die Logarithmus-Funktion ist die einfachste Funktion zur Definition eines Informationsgehalts mit diesen Eigenschaften.

Definition (Information und Entropie):

Information besteht in der Unsicherheit, Veränderungen eines Signals vorhersagen zu können. Der Informationsgehalt eines Zeichens hängt von der Wahrscheinlichkeit p ab, dass das informationstragende Signal zum Beobachtungszeitpunkt den diesem Zeichen zugeordneten Wert bzw. Wertebereich annimmt.
Der Informationsgehalt I eines Zeichens mit der Auftrittswahrscheinlichkeit p ist definiert als

$$I(p) = -\log_2 p \quad \text{mit} \quad [\,I\,] = \text{bit},$$

Der mittlere Informationsgehalt einer Quelle wird als Entropie H bezeichnet.

Beispiele:

- Deterministische (diskrete) Quelle, welche stets das Zeichen 'A' emittiert:

$$I(\Pr[X = A]) = I(1) = -\log_2 1\,\text{bit} = 0\,\text{bit}$$

Definition (Information und Entropie):

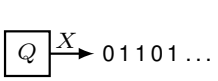
Information besteht in der Unsicherheit, Veränderungen eines Signals vorhersagen zu können. Der Informationsgehalt eines Zeichens hängt von der Wahrscheinlichkeit p ab, dass das informationstragende Signal zum Beobachtungszeitpunkt den diesem Zeichen zugeordneten Wert bzw. Wertebereich annimmt.
Der Informationsgehalt I eines Zeichens mit der Auftrittswahrscheinlichkeit p ist definiert als

$$I(p) = -\log_2 p \quad \text{mit} \quad [\,I\,] = \text{bit},$$

Der mittlere Informationsgehalt einer Quelle wird als Entropie H bezeichnet.

Beispiele:

- Binäre Quelle, welche auf nicht vorhersehbare Weise die Zeichen '0' oder '1' emittiert:



$$I(\Pr[X = 0]) = I(0.5) = -\log_2 0.5\,\text{bit} = 1\,\text{bit}$$
$$I(\Pr[X = 1]) = I(0.5) = -\log_2 0.5\,\text{bit} = 1\,\text{bit}$$

Die Entropie $H(X) = \sum_i p_i I(p_i)$ dieser Quelle beträgt

$$H(X) = -(p_0 \log_2(p_0) + p_1 \log_2(p_1)) = -(-0.5 - 0.5) = 1\text{bit/Zeichen},$$

Definition (Information und Entropie):

Information besteht in der Unsicherheit, Veränderungen eines Signals vorhersagen zu können. Der Informationsgehalt eines Zeichens hängt von der Wahrscheinlichkeit p ab, dass das informationstragende Signal zum Beobachtungszeitpunkt den diesem Zeichen zugeordneten Wert bzw. Wertebereich annimmt.
Der Informationsgehalt I eines Zeichens mit der Auftrittswahrscheinlichkeit p ist definiert als

$$I(p) = -\log_2 p \quad \text{mit} \quad [\,I\,] = \text{bit},$$

Der mittlere Informationsgehalt einer Quelle wird als Entropie H bezeichnet.

Beispiele:

- Ungeordnete Zeichen eines langen deutschen Textes, d. h. $X \in \{A, B, C, \ldots, Z\}$:

$$I(\Pr[X = E]) = I(0.1740) \approx 2.5223\,\text{bit}$$

Die Entropie $H(X)$ dieser Quelle beträgt

$$H(X) = -\sum_{i=1}^{N} p_i \log_2(p_i) \approx 4.0629\,\text{bit/Zeichen},$$

d. h. deutscher Text lässt sich mit durchschnittlich etwas mehr als 4 bit pro Zeichen kodieren. **Achtung:** Dies gilt nur für gedächtnislose Quellen. Andernfalls müssen bedingte Wahrscheinlichkeiten berücksichtigt werden!

Definition (Verbundentropie, bedingte Entropie):

Die Verbundentropie $H(X, Y)$ ist die Verallgemeinerung der Entropie für eine multivariate Zufallsvariable, d.h. für einen Zufallsvektor.
Der Zufallsvektor (X, Y) besitzt die Verbundentropie $H(X, Y)$.
Die bedingte Entropie $H(Y|X)$ zweier Zufallsvariablen X und Y ist die Unsicherheit über Y, die verbleibt, wenn X bereits bekannt ist.
Sind X und Y voneinander unabhängig, dann bleibt die Entropie von Y vollständig erhalten, wenn X bereits bekannt ist.
Sind X und Y voneinander abhängig, dann ist die bedingte Entropie kleiner als im unabhängigen Fall.

Berechnung der Verbundentropie und der bedingten Entropie:

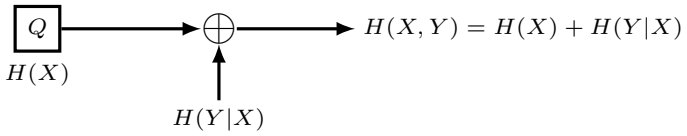
Die Verbundentropie $H(X, Y)$ entsteht aus der Addition der Quellenentropie $H(X)$ mit dem von dieser Quelle statistisch unabhängigen Anteil $H(Y|X)$ einer anderen Quelle:

$$H(X, Y) = H(X) + H(Y|X).$$

Die bedingte Entropie $H(Y|X)$ errechnet sich aus den Auftrittswahrscheinlichkeiten p_{XY} und p_Y:

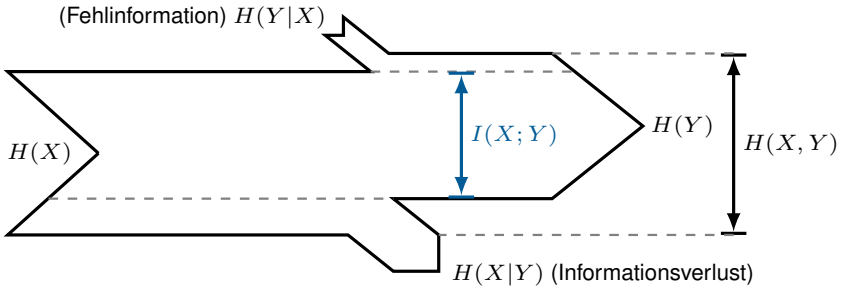
$$H(Y|X) = -\sum_{x\in\mathcal{X}}\sum_{y\in\mathcal{Y}} p_{XY}(X=x, Y=y)\log_2(p_Y(Y=y|X=x))$$

Beispiel: Hinzufügen von Fehlinformation $H(Y|X)$



- Die bedingte Entropie $H(Y|X)$ kann verstanden werden als die Unsicherheit in Y, wenn X bekannt ist.
- Da die Fehlinformation keine Nutzinformation beisteuert, kann die Verbundentropie auf der Empfangsseite $H(X,Y)$ nicht der Nutzinformation entsprechen.

Informationstheoretisches Modell eines gedächtnislosen Kanals:



- Für die Entropie auf der Empfangsseite (Empfangsentropie) erhalten wir $H(Y) = H(X) - H(X|Y) + H(Y|X)$ (Sendeentropie abzüglich eines Informationsverlusts durch den Kanal zuzüglich der Fehlinformation).
- Die transportierte Information (Transinformation) entspricht der Sendeentropie abzüglich des Informationsverlusts bzw. der Empfangsentropie abzüglich der Fehlinformation.

Definition (Transinformation):

Die von Sender zu Empfänger über einen gedächtnislosen Kanal transportierte Information bezeichnet man als Transinformation (engl. Mutual Information)

$$I(X;Y) = H(X) - H(X|Y) = H(Y) - H(Y|X).$$

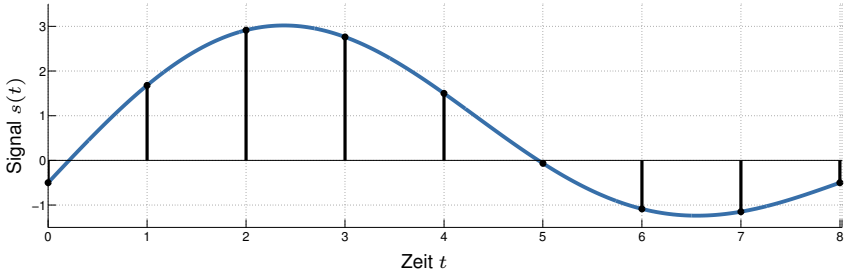
Welche Bedeutung hat ein bestimmtes Signal?:

Ein Signal transportiert Information. Erst durch eine Interpretationsvorschrift erhält diese Information eine Bedeutung, d. h. es muss eine Abbildung zwischen Symbolen (Signalwerten bzw. Wertebereichen) und Daten geben.

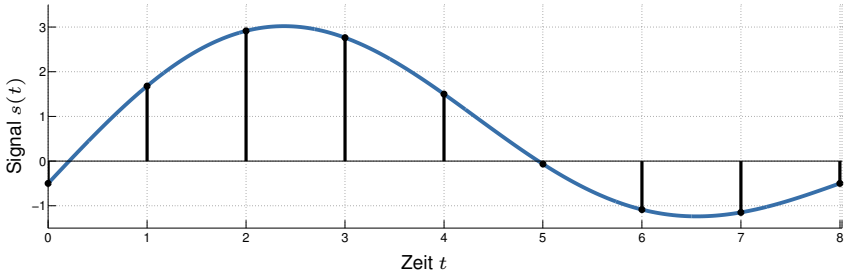
Beispiel: Gegeben sei ein binäres Alphabet mit den Zeichen $X \in \{0, 1\}$. Die Interpretationsvorschrift laute

$$x = \begin{cases} 0 & s(t) \leq 0, \\ 1 & \text{sonst.} \end{cases}$$

Welche Bedeutung hat das unten abgebildete Signal?



Offene Fragen



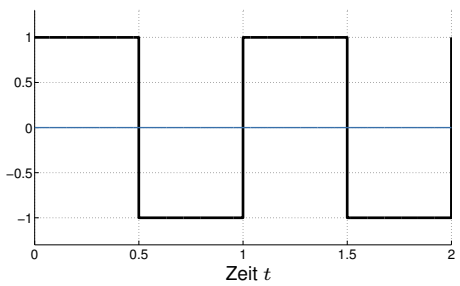
- In welchen zeitlichen Abständen werden Samples genommen? (Zeitdiskretisierung)
- Bedeutet häufigeres Abtasten auch automatisch mehr Information? (Abtasttheorem)
- Wie werden kontinuierliche Signalwerte gerundet? (Quantisierung)
- Welche Abbildungs- / Interpretationsvorschriften gibt es? (Leitungskodierung)
- Welche Störfaktoren spielen eine Rolle? (Rauschen, Dämpfung, Verzerrung, ...)
- Wie werden Fehler erkannt und ggf. korrigiert? (Kanalkodierung)
- Und wie wird ein derartiges Signal überhaupt erzeugt? (Impulsformung, Modulation)

1.2. Klassifizierung von Signalen

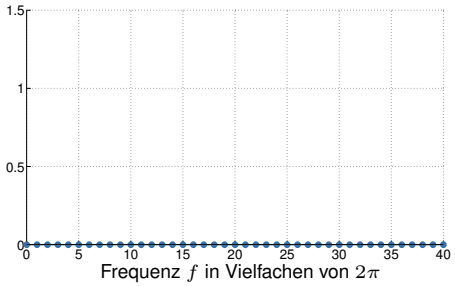
1.2.1. Zeit- und Frequenzbereich

1.2.1.1. Zeit- und Frequenzbereich

(Periodische) Zeitsignale lassen sich als Überlagerung von Sinus- und Kosinusschwingungen unterschiedlicher Frequenzen auffassen: **Fourier-Reihe:** (mit $\omega = 2\pi$)



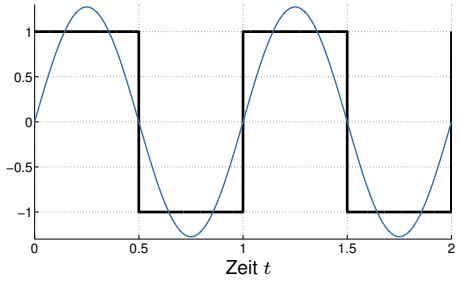
(a) Zeitsignal $s(t)$



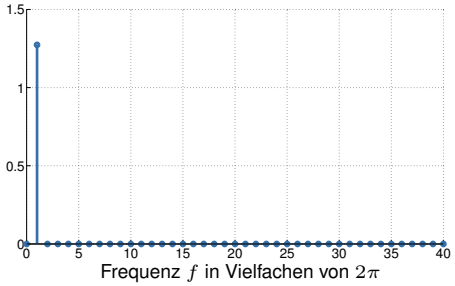
(b) Spektrum $S(f)$

$$s(t) \approx \frac{a_0}{2}$$

(Periodische) Zeitsignale lassen sich als Überlagerung von Sinus- und Kosinusschwingungen unterschiedlicher Frequenzen auffassen: **Fourier-Reihe:** (mit $\omega = 2\pi$)



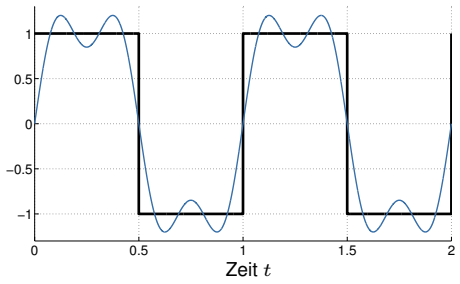
(a) Zeitsignal $s(t)$



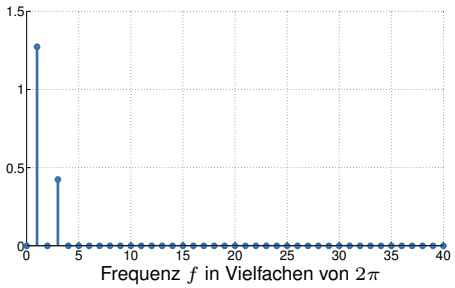
(b) Spektrum $S(f)$

$$s(t) \approx \frac{a_0}{2} + a_1 \cos(\omega t) + b_1 \sin(\omega t)$$

(Periodische) Zeitsignale lassen sich als Überlagerung von Sinus- und Kosinusschwingungen unterschiedlicher Frequenzen auffassen: **Fourier-Reihe:** (mit $\omega = 2\pi$)



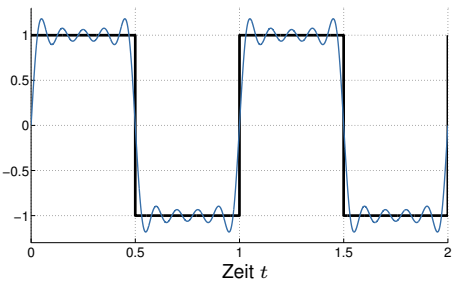
(a) Zeitsignal $s(t)$



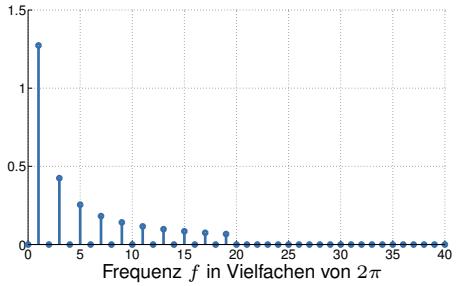
(b) Spektrum $S(f)$

$$s(t) \approx \frac{a_0}{2} + \sum_{k=1}^{4} (a_k \cos(k\omega t) + b_k \sin(k\omega t))$$

(Periodische) Zeitsignale lassen sich als Überlagerung von Sinus- und Kosinusschwingungen unterschiedlicher Frequenzen auffassen: **Fourier-Reihe:** (mit $\omega = 2\pi$)



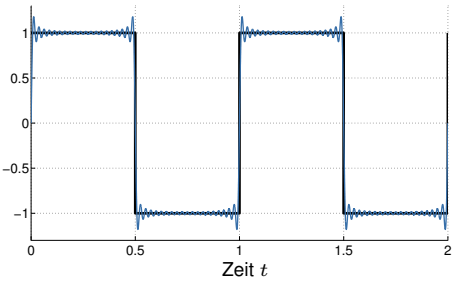
(a) Zeitsignal $s(t)$



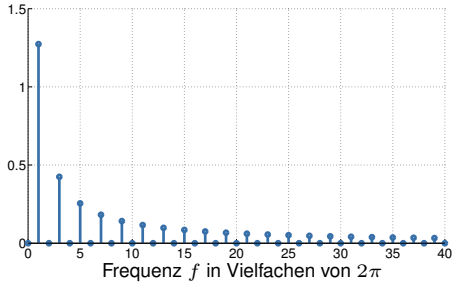
(b) Spektrum $S(f)$

$$s(t) \approx \frac{a_0}{2} + \sum_{k=1}^{19} (a_k \cos(k\omega t) + b_k \sin(k\omega t))$$

(Periodische) Zeitsignale lassen sich als Überlagerung von Sinus- und Kosinusschwingungen unterschiedlicher Frequenzen auffassen: **Fourier-Reihe:** (mit $\omega = 2\pi$)



(a) Zeitsignal $s(t)$



(b) Spektrum $S(f)$

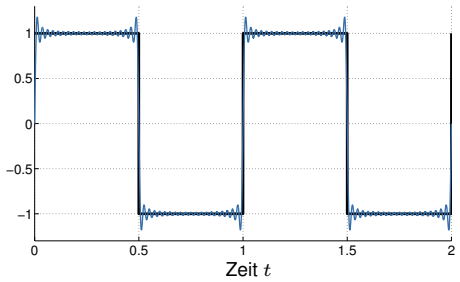
$$s(t) \approx \frac{a_0}{2} + \sum_{k=1}^{39} (a_k \cos(k\omega t) + b_k \sin(k\omega t))$$

(Periodische) Zeitsignale lassen sich als Überlagerung von Sinus- und Kosinusschwingungen unterschiedlicher Frequenzen auffassen: **Fourier-Reihe:** (mit $\omega = 2\pi$)

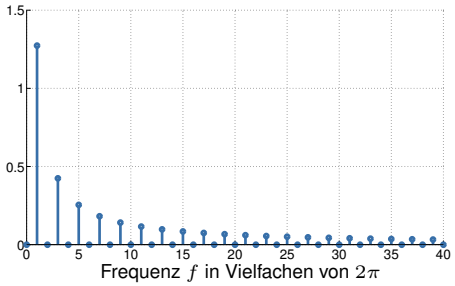
:

Im Grenzwert $N \to \infty$ gilt:

$$s(t) = \frac{a_0}{2} + \sum_{k=1}^{\infty} (a_k \cos(k\omega t) + b_k \sin(k\omega t))$$



(a) Zeitsignal $s(t)$



(b) Spektrum $S(f)$

Fourierreihe:

Eine beliebige periodische Funktion $s(t)$ lässt sich als Summe gewichteter Sinus- und Kosinus-Schwingungen darstellen. Die so entstehende Reihenentwichlung von $s(t)$ bezeichnet man als Fourierreihe:

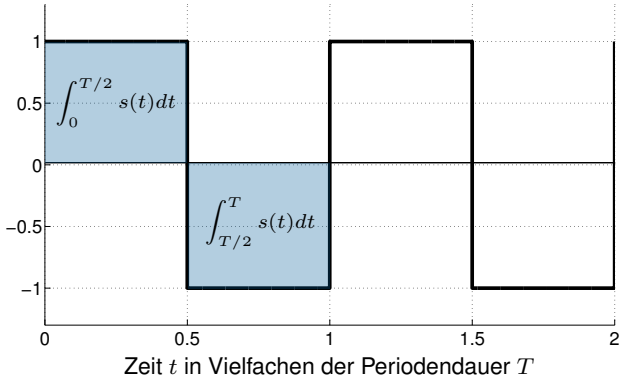
$$s(t) = \frac{a_0}{2} + \sum_{k=1}^{\infty} \left(a_k \cos(k\omega t) + b_k \sin(k\omega t) \right). \tag{1.1}$$

Das k-te Summenglied bezeichnet man auch als k-te Harmonische. Das konstante Glied $a_0/2$ repräsentiert eine Verschiebung der Signalamplitude bezüglich der Ordinate (y-Achse) und damit den konstanten Anteil der Funktion. Die Koeffizienten a_k und b_k lassen sich wie folgt bestimmen:

$$a_k = \frac{2}{T} \int_0^T s(t) \cdot \cos(k\omega t) dt \quad \text{und} \quad b_k = \frac{2}{T} \int_0^T s(t) \cdot \sin(k\omega t) dt. \tag{1.2}$$

Einfache Signaleigenschaften „by inspection“

- Die Bestimmung der Koeffizienten a_k und b_k ist nur Rechenarbeit
- Einige Eigenschaften sieht man einfachen Signalen aber leicht an, z. B.:

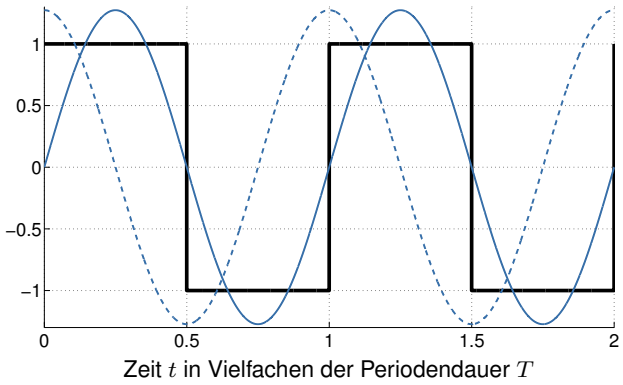


- Punktsymmetrie zu $(T/2, 0) \quad \Rightarrow \quad a_0 = \frac{2}{T} \int_0^T s(t)\ dt = 0$

- Kein Gleichanteil (d.h. keine Verschiebung entlang der Ordinate)

Frage: Was ist mit dem Signal $s'(t) = s(t) + c$ mit $c > 0$? **Einfache Signaleigenschaften „by inspection“**

- Die Bestimmung der Koeffizienten a_k und b_k ist nur Rechenarbeit
- Einige Eigenschaften sieht man einfachen Signalen aber leicht an, z. B.:



- Alle Gewichte für den Kosinus sind null, also $a_k = 0\ \forall k \in \mathbb{N}$
- Grund: $s(t)$ ist genau in Phase mit dem Sinus

Frage: Was ist, wenn man die Phase von $s(t)$ um 90° verschiebt?
Bislang haben wir nur periodische Signale betrachtet. Was ist mit nicht-periodischen Signalen?

- Keine Entwicklung als Fourierreihe möglich
- Kontinuierliches (anstatt diskretes) Spektrum
- Frequenzbereich erhält man mittels Fouriertransformation

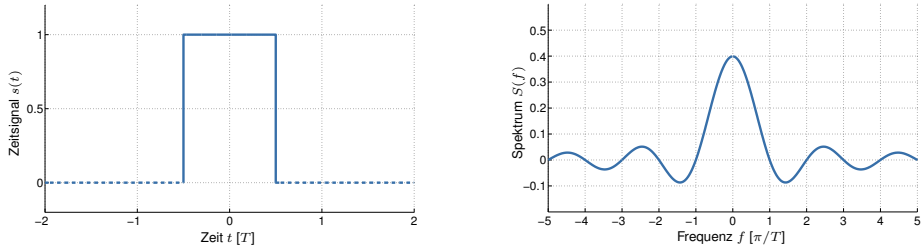
Fouriertransformation:

Die Fourier-Transformierte einer stetigen, integrierbaren Funktion $s(t)$ ist gegeben als

$$s(t) \quad \circ\!\!-\!\!\bullet \quad S(f) = \mathcal{F}(s) = \frac{1}{\sqrt{2\pi}} \int_{t=-\infty}^{\infty} s(t) e^{-i2\pi f t}\, dt.$$

Beispiel: Rechteckimpuls und zugehöriges Spektrum

$$\mathrm{rect}(t) \circ\!\!-\!\!\bullet \frac{1}{\sqrt{2\pi}T} \mathrm{sinc}\left(\frac{f}{T}\right), \quad \text{mit } \mathrm{sinc}(x) = \frac{\sin(\pi x)}{\pi x}$$



1.2.2. Abtastung, Rekonstruktion und Quantisierung

1.2.2.1. Abtastung, Rekonstruktion und Quantisierung [14]

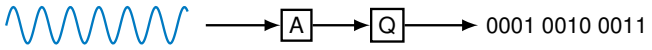
Natürlich vorkommende Signale sind zeitkontinuierlich und wertkontinuierlich, d. h. sie nehmen zu unendlich vielen Zeitpunkten beliebige reelle Werte an. **Problem für Computer:**

- Endlicher Speicher
- Endliche Rechengenauigkeit

Lösung: Diskretisierung von Signalen im

- Zeitbereich (Abtastung) und
- Wertbereich (Quantisierung).

Ein zeit- und wertdiskretes Signal ist digital und wird in Wörtern fester Länge gespeichert.



Vergleiche: Nutzung von Fix- bzw. Gleitkommazahlen anstelle von reellen Zahlen entspricht einer Rundung (Quantisierung) auf eine endliche Anzahl diskreter Stufen.
Abtastung Das Signal $s(t)$ wird mittels des Einheitsimpulses (Dirac-Impulses) $\delta[t]$ in äquidistanten Abständen T_a (Abtastintervall) für $n \in \mathbb{Z}$ abgetastet:

$$\hat{s}(t) = s(t) \sum_{n=-\infty}^{\infty} \delta[t - nT_a], \text{ mit } \delta[t - nT_a] = \begin{cases} 1 & t = nT_a, \\ 0 & \text{sonst.} \end{cases}$$

Da $\hat{s}(t)$ nur zu den Zeitpunkten nT_a für ganzzahlige n von Null verschieden ist, vereinbaren wir die Schreibweise $\hat{s}[n]$ für zeitdiskrete aber wertkontinuierliche Signale. **Abtastung** Das Signal $s(t)$ wird mittels des Einheitsimpulses (Dirac-Impulses) $\delta[t]$ in äquidistanten Abständen T_a (Abtastintervall) für $n \in \mathbb{Z}$ abgetastet:

$$\hat{s}(t) = s(t) \sum_{n=-\infty}^{\infty} \delta[t - nT_a], \text{ mit } \delta[t - nT_a] = \begin{cases} 1 & t = nT_a, \\ 0 & \text{sonst.} \end{cases}$$

Da $\hat{s}(t)$ nur zu den Zeitpunkten nT_a für ganzzahlige n von Null verschieden ist, vereinbaren wir die Schreibweise $\hat{s}[n]$ für zeitdiskrete aber wertkontinuierliche Signale.
Rekonstruktion Mittels der Abtastwerte $\hat{s}[n]$ ist es möglich, das ursprüngliche Signal $s(t)$ zu rekonstruieren:

$$s(t) \approx \sum_{n=-\infty}^{\infty} \hat{s}[n] \cdot \operatorname{sinc}\left(\frac{t - nT_a}{T_a}\right).$$

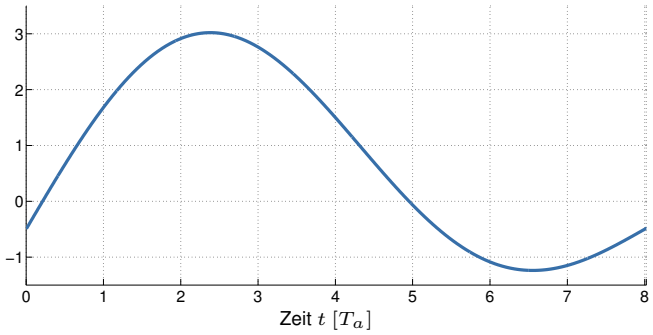


Abbildung 1.2.: Zeitkontinuierliches Signal $s(t)$

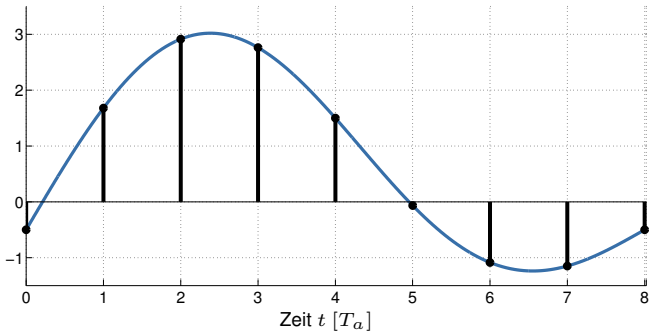


Abbildung 1.3.: Zeitkontinuierliches Signal $s(t)$ und abgetastetes Signal $\hat{s}[n]$

- Abtastwerte sind Stützstellen und
- dienen als Gewichte für eine passende Ansatzfunktion
  (trigonometrische Interpolation, vgl. Polynominterpolation → Numerisches Programmieren).

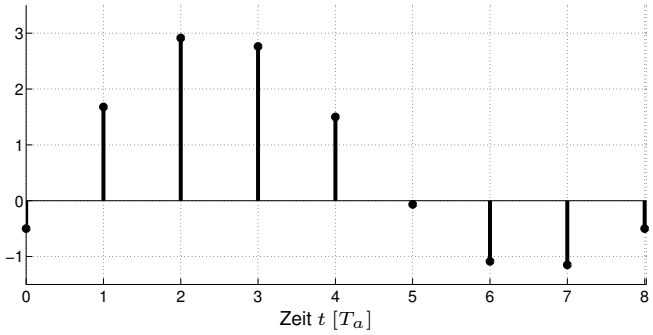


Abbildung 1.4.: Abgetastetes Signal $\hat{s}[n]$

Rekonstruktion Mittels der Abtastwerte $\hat{s}[n]$ ist es möglich, das ursprüngliche Signal $s(t)$ zu rekonstruieren:

$$s(t) \approx \sum_{n=-\infty}^{\infty} \hat{s}[n] \cdot \operatorname{sinc}\left(\frac{t - nT_a}{T_a}\right).$$

- Abtastwerte sind Stützstellen und
- dienen als Gewichte für eine passende Ansatzfunktion (trigonometrische Interpolation, vgl. Polynominterpolation → Numerisches Programmieren).

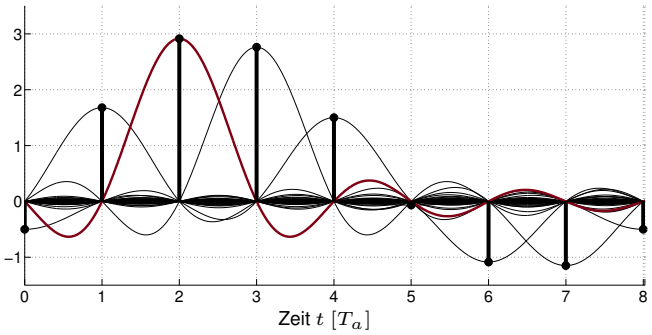


Abbildung 1.5.: Abtastwerte dienen als Gewichte für eine Reihe zeitlich verschobener Ansatzfunktionen

Rekonstruktion Mittels der Abtastwerte $\hat{s}[n]$ ist es möglich, das ursprüngliche Signal $s(t)$ zu rekonstruieren:

$$s(t) \approx \sum_{n=-\infty}^{\infty} \hat{s}[n] \cdot \operatorname{sinc}\left(\frac{t - nT_a}{T_a}\right).$$

- Abtastwerte sind Stützstellen und
- dienen als Gewichte für eine passende Ansatzfunktion (trigonometrische Interpolation, vgl. Polynominterpolation → Numerisches Programmieren).

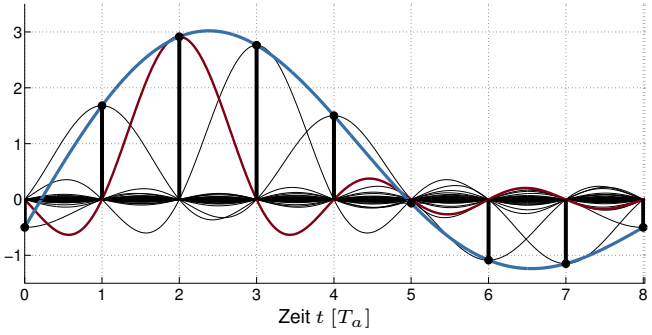


Abbildung 1.6.: Die Summe der gewichteten Ansatzfunktion rekonstruiert bzw. approximiert $s(t)$

Wann ist eine verlustfreie Rekonstruktion möglich?

- Die Multiplikation im Zeitbereich entspricht einer Faltung im Frequenzbereich:

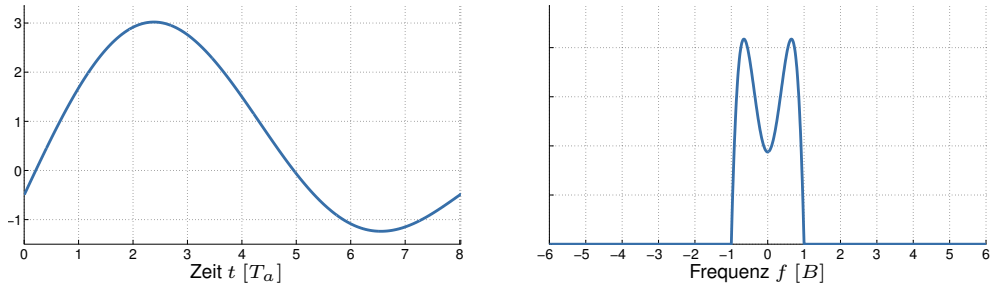
$$s(t) \cdot \delta[t - nT] \quad \circ\!\!-\!\!\bullet \quad \frac{1}{T} S(f) * \delta[f - n/T].$$

- Diese Faltung mit Einheitsimpulsen entspricht einer Verschiebung von $S(f)$ entlang der Abszisse.

:

Folglich entspricht die Abtastung des Signals $s(t)$ in Abständen T_a der periodischen Wiederholung seines Spektrums $S(f)$ in Abständen von $f_a = 1/T_a$.

Beispiel: Abtastung eines auf B bandbegrenzten Signals $s(t)$ mit der Abtastfrequenz $f_a = 3B$ **Wann ist eine verlustfreie Rekonstruktion möglich?**



(a) Ursprüngliches Signal $s(t)$

(b) Zugehöriges Spektrum $S(f)$ (schematisch)

- Die Multiplikation im Zeitbereich entspricht einer Faltung im Frequenzbereich:

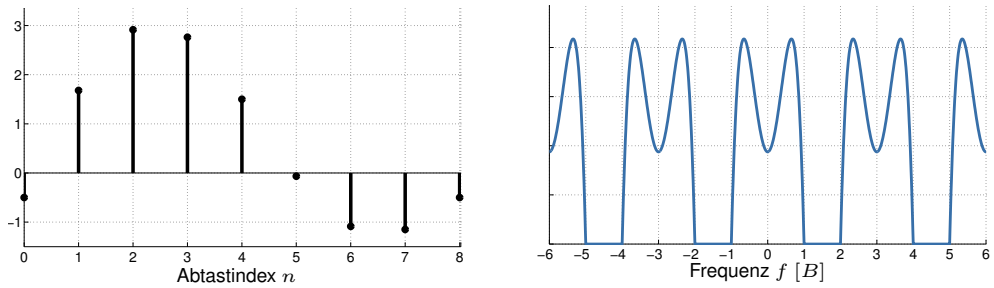
$$s(t) \cdot \delta[t - nT] \quad \circ\!\!-\!\!\bullet \quad \frac{1}{T} S(f) * \delta[f - n/T].$$

- Diese Faltung mit Einheitsimpulsen entspricht einer Verschiebung von $S(f)$ entlang der Abszisse.

:

Folglich entspricht die Abtastung des Signals $s(t)$ in Abständen T_a der periodischen Wiederholung seines Spektrums $S(f)$ in Abständen von $f_a = 1/T_a$.

Beispiel: Abtastung eines auf B bandbegrenzten Signals $s(t)$ mit der Abtastfrequenz $f_a = 3B$



(a) Abgetastetes Signal $\hat{s}[n]$

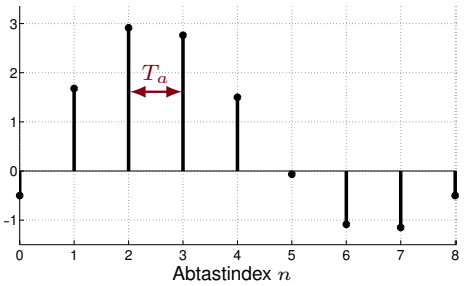
(b) Zugehöriges Spektrum $\hat{S}(f)$ (schematisch)

Abtasttheorem von Shannon und Nyquist:

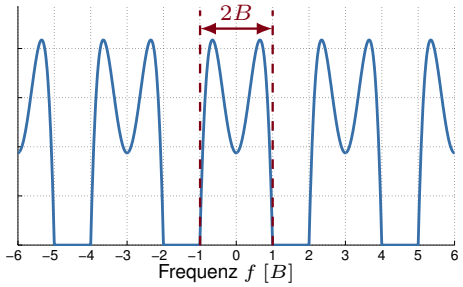
Ein auf $|f| \leq B$ bandbegrenztes Signal $s(t)$ ist vollständig durch äquidistante Abtastwerte $\hat{s}[n]$ beschrieben, sofern diese nicht weiter als $T_a = 1/2B$ auseinander liegen. Die Abtastfrequenz, welche eine vollständige Signalrekonstruktion erlaubt, ist folglich durch

$$f_a > 2B$$

nach unten beschränkt.



(a) Abgetastetes Signal $\hat{s}[n]$



(b) Zugehöriges Spektrum $\hat{S}(f)$ (schematisch)

- Wählt man $f_a < 2B$, so überlappen sich die periodischen Wiederholungen des Spektrums
- Diesen Effekt bezeichnet man als Aliasing
- Eine verlustfreie Rekonstruktion ist in diesem Fall nicht möglich

Quantisierung

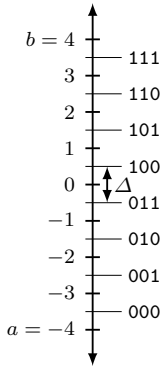
- Bei N bit Wortbreite sind $M = 2^N$ diskrete Signalstufen unterscheidbar
- Diese werden in einem Quantisierungsintervall $I_Q = [a, b]$ „sinnvoll“ verteilt

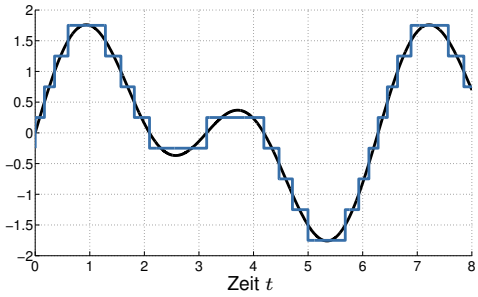
Beispiel: Lineare Quantisierung mit mathematischem Runden

- Optimal, wenn alle Werte innerhalb I_Q mit gleicher Wahrscheinlichkeit auftreten
- Stufenbreite $\Delta = \dfrac{b-a}{M}$
- Innerhalb I_Q beträgt der maximale Quantisierungsfehler $q_{\max} = \Delta/2$
- Signalwerte außerhalb I_Q werden auf obere bzw. untere Grenze von I_Q normiert $\Rightarrow$ Außerhalb I_Q ist der Quantisierungsfehler unbeschränkt

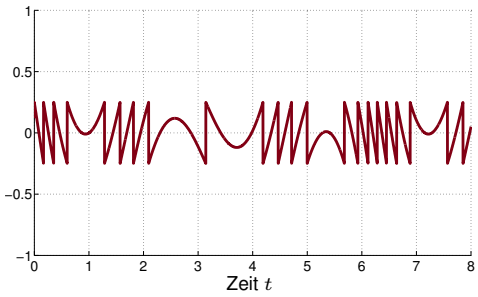
Was ist, wenn die Signalwerte nicht gleichverteilt sind?

- Lineare Quantisierung ist typischerweise suboptimal
- Nicht-lineare Quantisierung wird beispielsweise bei der Digitalisierung von Sprache oder Musik eingesetzt.





(a) Quantisiertes Signal $\overline{s}(t)$ (blau)



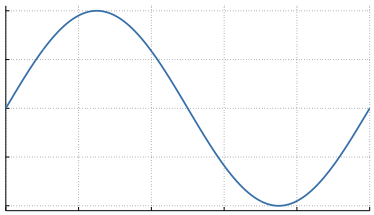
(b) Quantisierungsfehler $q_e(t) = s(t) - \overline{s}(t)$

Quantisierung (Beispiel) Lineare Quantisierung im Intervall $I = [-2, 2]$ mit $N = 3$ bit:

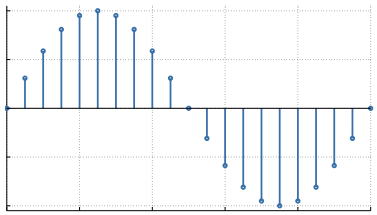
Codewort	000	001	010	011	100	101	110	111
Signalstufe	$-7/4$	$-5/4$	$-3/4$	$-1/4$	$1/4$	$3/4$	$5/4$	$7/4$

- Die Zuweisung von Codewörtern zu Signalstufen ist im Prinzip willkürlich
- Häufig wählt man jedoch einen Code, welcher die Auswirkung einzelner Bitfehler reduziert
  (z. B. Gray-Code: Benachbarte Codewörter unterscheiden sich nur in jeweils einer binären Ziffer, d.h. die Hamming-Distanz ist 1.)

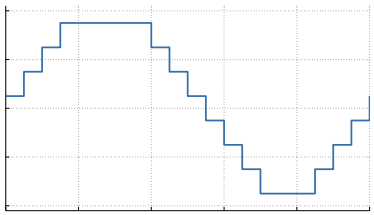
Übersicht: Signaltypen



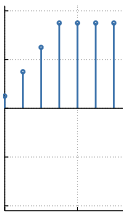
(a) Analog $s(t)$



(b) Zeitdiskret und wertkontinuierlich $\hat{s}[n]$



(c) Zeitkontinuierlich und wertdiskret $\overline{s}(t)$



1.3. Übertragungskanal

Aus dem letzten Teilkapitel sollten wir wissen:

- Was sind die Unterschiede zwischen analogen, zeitdiskreten, wertdiskreten und digitalen Signalen?
- Wie muss ein Signal abgetastet werden, so dass keine Information verloren geht?
- Unter welchen Bedingungen kann ein natürlich vorkommendes Signal aus abgetasteten und quantisierten Werten verlustfrei rekonstruiert werden?
- Wie sollten die Abtastwerte quantisiert werden, wenn innerhalb des Quantisierungsintervalls jeder Signalpegel gleich wahrscheinlich ist?

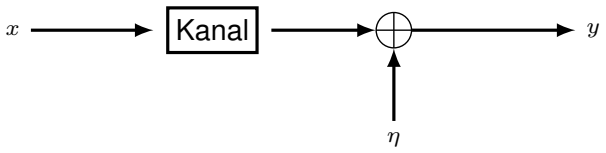
In diesem Abschnitt klären wir die folgenden Fragen:

- Welchen Einfluss hat der Übertragungskanal auf ein Signal?
- Wie hoch ist die theoretisch maximal erzielbare Übertragungsrate?

1.3.1. Kanaleinflüsse

1.3.1.1. Kanaleinflüsse

Modellvorstellung eines (linearen, zeitinvarianten) Kanals mit einem Ein- und Ausgang:



Unser Modell berücksichtigt:

- Dämpfung (Signalamplitude des Nutzsignals am Ausgang ist geringer als am Eingang)
- Tiefpassfilterung (höhere Frequenzen werden stärker gedämpft als niedrige)
- Verzögerung (die Übertragung benötigt eine gewisse Zeit)
- Rauschen in Form von Additive White Gaussian Noise (AWGN)[1]

Additive bedeutet, dass das Rauschen zum Signal hinzuaddiert wird (in Abgrenzung gegen Rauschen, welche multiplikativ mit dem Signal verknüpft wird). Weiß bezieht sich darauf, dass im Rauschen Anteile aller mögliche Frequenzen auftreten. Farbiges Rauschen beinhaltet nur Rauschen in den "zur Farbe gehörigen" Frequenzanteilen. **Wir berücksichtigen nicht (weil zu kompliziert):**

- Phasenverzerrung, d.h. frequenzabhängige Phasenverschiebungen
- Interferenzen durch andere Übertragungen
- Reflektionen eigener Signale
- Zeitvariante Einflüsse, z. B. können Objekte und Personen eine Funkübertragung beeinflussen

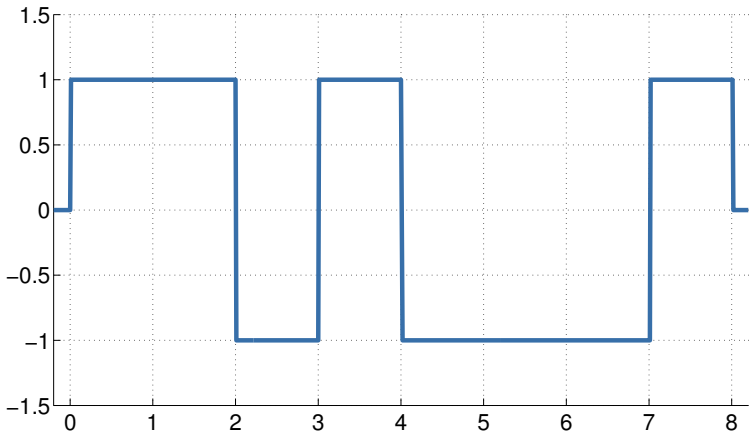


Abbildung 1.7.: Idealisiertes Sendesignal

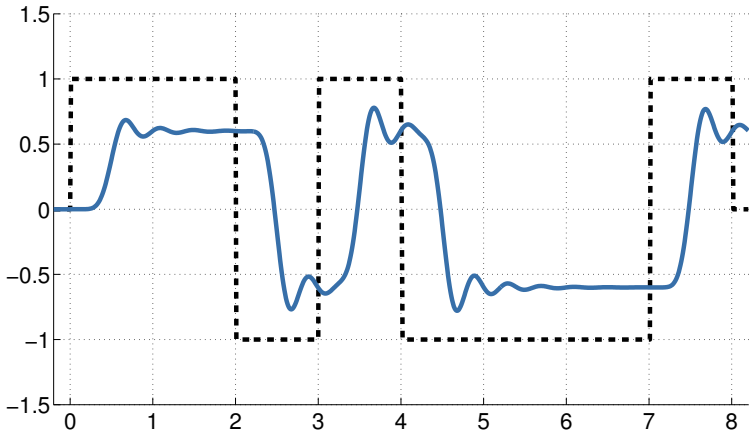


Abbildung 1.8.: Sendesignal nach Dämpfung und Tiefpasseinflüssen durch den Kanal

Beispiel: Die Dämpfung verursacht die Verringerung der Amplitude. Durch den Tiefpasseinfluss gehen steile Flanken verloren, sodass die Kurve runder wird. **Beispiel:** Die Dämpfung verursacht die Verringerung der Amplitude. Durch den Tiefpasseinfluss gehen steile Flanken verloren, sodass die Kurve runder wird. **Beispiel:** Die Dämpfung verursacht die Verringerung der Amplitude. Durch den Tiefpasseinfluss gehen steile Flanken verloren, sodass die Kurve runder wird.

1.3.2. Kanalkapazität

1.3.2.1. Kanalkapazität

Wir haben bereits gesehen, dass

- ein Kanal wie ein Tiefpass wirkt und
- zusätzliches Rauschen die Übertragung stört.

[1] AWGN ist eine vereinfachende Modellvorstellung von Rauschprozessen. In der Realität gibt es kein AWGN.

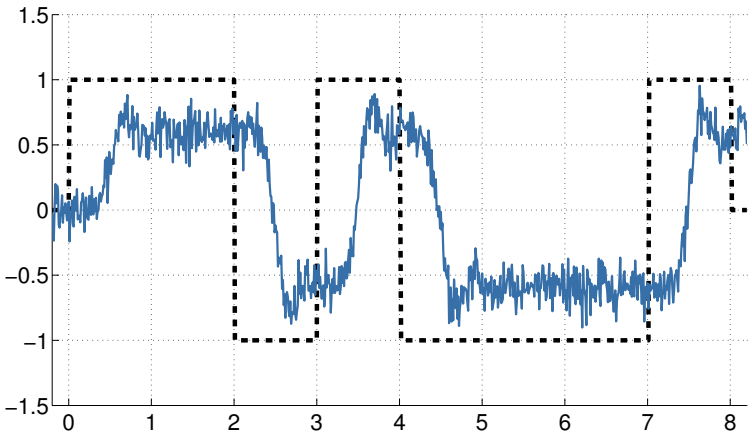


Abbildung 1.9.: Sendesignal nach Dämpfung und Tiefpasseinflüssen durch den Kanal sowie mit AWGN

Wegen der für Kanäle typischen Tiefpasscharakteristik kann man von einer Kanalbandbreite B sprechen:

- Niedrige Frequenzen passieren ungehindert (Tiefpass)
- Hohe Frequenzen werden gedämpft
- Ab einer bestimmten Frequenz ist die Dämpfung so stark, dass die betreffenden Signalanteile vernachlässigt werden können

Vereinfacht nehmen wir eine scharfe Grenze für B an:

- Frequenzanteile $|f| < B$ passieren
- Frequenzanteile $|f| \geq B$ werden gesperrt

Wie hoch ist die erzielbare Datenrate auf einem Kanal mit Bandbreite B?:

Hierfür benötigen wir einen Zusammenhang zwischen

- der Kanalbandbreite B,
- der Anzahl M unterscheidbarer Signalstufen und
- dem Verhältnis zwischen der Leistung des Nutzsignals und des Rauschens.

Rauschfreier, binärer Kanal Wir erinnern uns an das Abtasttheorem:

- Ein auf B bandbegrenztes Signal muss
  mind. mit der Frequenz $2B$ abgetastet werden,

so dass das Signal verlustfrei rekonstruiert werden kann, d. h. dass keine Information verloren geht. Anders herum betrachtet: Man erhält aus einem Signal

- bis zu $2B$ unterscheidbare[2] und
  voneinander unabhängige Werte.

[2] Hinreichend empfindliche Messsysteme vorausgesetzt

Tastet man häufiger ab, gewinnt man keine neue Information. Dies führt zu einer neuen Interpretation der Frequenz $f = 2B$, welche auch als Nyquist-Rate bezeichnet wird.

Definition: Nyquist-Rate:

Sei B die Grenzfrequenz eines bandbegrenzten Kanals. Dann ist die Nyquist-Rate $f_N = 2B$

- eine untere Schranke für die minimale Abtastrate, die eine vollständige Rekonstruktion des Signals erlaubt,
- eine obere Schranke für die Anzahl an Werten je Zeiteinheit, die nach der Übertragung über den Kanal unterscheidbar und unabhängig voneinander sind.

Rauschfreier, M-ärer Kanal Angenommen es können nicht nur zwei sondern $M = 2^N$ unterscheidbare Symbole übertragen werden. Wie ändert sich die erzielbare Datenrate? Wir erinnern uns an Quantisierung und Entropie:

- Mit einer Wortbreite von N bit lassen sich $M = 2^N$ diskrete Signalstufen darstellen.
- Emittiert eine Quelle alle Zeichen (Signalstufen) mit der gleichen Wahrscheinlichkeit, so ist die Entropie (und damit die mittlere Information) der Quelle maximal.

Folglich erhalten wir für die Übertragungsrate

- über einen Kanal der Bandbreite B
- die maximale Rate $R_{\max} = 2B \log_2(M)$ bit.

Hartleys Gesetz:

Auf einem Kanal der Bandbreite B mit M unterscheidbaren Signalstufen ist die Kanalkapazität durch

$$C_H = 2B \log_2(M) \text{ bit}$$

nach oben begrenzt.

Interessant: Wenn wir beliebig viele Signalstufen voneinander unterscheiden könnten, wäre die erzielbare Datenrate unbegrenzt! Wo ist das Problem?
Rauschen

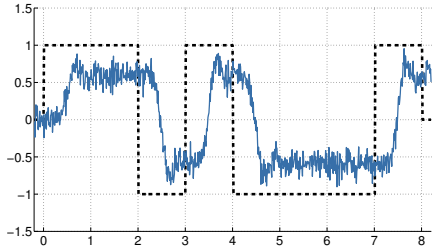
- Rauschen macht es schwer, Signalstufen auseinanderzuhalten
- Je feiner die Signalstufen gewählt werden, desto schwieriger wird dies

Maß für die Stärke des Rauschens:

$$\text{SNR} = \frac{\text{Signalleistung}}{\text{Rauschleistung}} = \frac{P_S}{P_N}$$

Das Signal to Noise Ratio (SNR) wird in der Einheit dB angegeben:

$$\text{SNR dB} = 10 \cdot \log_{10}(\text{SNR})$$



Beispiel: $P_S = 1\,\text{mW},\ P_N = 0.5\,\text{mW}$

$$\text{SNR} = 10 \cdot \log_{10}\left(\frac{1}{0.5}\right)\,\text{dB} \approx 3.0\,\text{dB}$$

Signalleistung

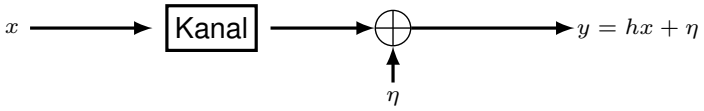
Definition (Leistung eines Signals):

Der Mittelwert über das Quadrat eines Signals ist der Mittelwert der Leistung. Dieser Mittelwert wird Wirkleistung genannt.
Die Varianz (Streuung) einer Zufallsvariable entspricht der Signalleistung ohne deren Gleichanteil und stellt die informationstragende Leistung eines Signals dar.

Die Signalleistung erwartungswertfreier (gleichanteilsfreier) Signale beträgt: $\text{Var}[y] = \int_{-\infty}^{\infty} y^2 p(y)\,dy = P_y$.

Rauschbehafteter, M-ärer Kanal

- Keine Quantisierungsfehler (da wir nur an der Rausch-Abhängigkeit interessiert sind)
- Das (erwartungswertfreie) Sendesignal x werde durch den Kanal um den Faktor $h < 1$ gedämpft und von (erwartungswertfreiem) unabhängigem additivem Rauschen η überlagert



Für die Signalleistung[3] P_y am Kanalausgang erhalten wir

$$P_y = \text{Var}[y] = \text{Var}[hx + \eta] = h^2\text{Var}[x] + \text{Var}[\eta].$$

Für den Spezialfall, dass $\eta \sim \mathcal{N}(0, \sigma_\eta^2)$ (Gaussian Channel), lässt sich zeigen, dass ein ebenfalls normalverteiltes $x \sim \mathcal{N}(0, \sigma_x^2)$ die Datenrate maximiert. Wir erhalten in diesem Fall

$$P_y = h^2\sigma_x^2 + \sigma_\eta^2.$$

[3] erwartungswertfreie ergodische Signale: $\text{Var}[y] = \int_{-\infty}^{\infty} y^2 p(y)\,dy = \lim_{T\to\infty} \frac{1}{T}\int_{-T/2}^{T/2} y^2(t)\,dt = P_y$.

Setzt man nun P_y ins Verhältnis zur Rauschleistung $P_N = \sigma_\eta^2$, so erhalten wir

$$\frac{h^2\sigma_x^2 + \sigma_\eta^2}{\sigma_\eta^2} = 1 + \frac{h^2\sigma_x^2}{\sigma_\eta^2} = 1 + \frac{P_S}{P_N} = 1 + \text{SNR}.$$

P_S ist die Leistung des Nutzsignals, die beim Empfänger ankommt.

Shannon-Hartley-Theorem:

Auf einem Kanal der Bandbreite B mit additiven weißen Rauschen mit Rauschleistung P_N und Signalleistung P_S beträgt die obere Schranke für die erreichbare Datenrate

$$C_S = B \log_2 \left(1 + \frac{P_S}{P_N}\right) \text{ bit}.$$

Herleitung des Theorems: siehe Shannons Veröffentlichung Communication in the Presence of Noise von 1949. **Vergleich mit Hartleys Gesetz (aufschlussreich!):**

$$C_H = 2B \log_2(M) = 2B \log_2 \left(\frac{b-a}{\Delta}\right).$$

- Die Intervallgrenzen a, b beziehen sich hier auf das unquantisierte Signal
- Mit $\alpha = a + \Delta/2$ und $\beta = b - \Delta/2$ als minimale bzw. maximale quantisierte Signalamplitude erhalten wir

$$C_H = 2B \log_2 \left(\frac{\beta - \alpha + \Delta}{\Delta}\right) = B \log_2 \left(\left(1 + \frac{\beta - \alpha}{\Delta}\right)^2\right). \tag{1.3}$$

Wird (1.3) ausmultipliziert, kommt aber etwas anderes raus!

- C_S **berücksichtigt nur additives Rauschen des Kanals, aber keine Quantisierungsfehler.**
- C_H **berücksichtigt nur die Signalstufen und damit die Quantisierung („Quantisierungsrauschen"), aber keine Kanaleinflüsse.**
- Der fehlende gemischte Term, wenn man (1.3) ausmultipliziert und mit C_S vergleicht, liegt in der Unabhängigkeitsannahme des Nutzsignals und des Rauschens begründet ($\mathrm{E}[x\eta] = \mathrm{E}[x]\mathrm{E}[\eta]$).
  Der Quantisierungsfehler ist natürlich nicht unabhängig vom Eingangssignal – aus diesem Grund lässt sich (1.3) nicht ohne Näherung in die selbe Form wie C_S bringen.

Zusammenfassung Die Kanalkapazität C ist durch zwei Faktoren beschränkt:

- **Die Anzahl M der unterscheidbaren Symbole**
  Selbst ein rauschfreier Kanal hilft nichts, wenn wir nur zwei Symbole nutzen (können).
- **Signal-to-Noise Ratio (SNR)**
  Ist das Signal-zu-Rausch-Verhältnis SNR zu gering, muss ggf. der Abstand Δ zwischen den Signalstufen erhöht und damit die Anzahl unterscheidbarer Symbole verringert werden, um eine zuverlässige Unterscheidung gewährleisten zu können.

:

Für die tatsächliche Kanalkapazität C gilt also folgende obere Schranke:

$$C < \min\{C_H, C_S\} = \min\left\{2B \log_2(M), B \log_2(1 + \text{SNR})\right\} \text{ bit}.$$

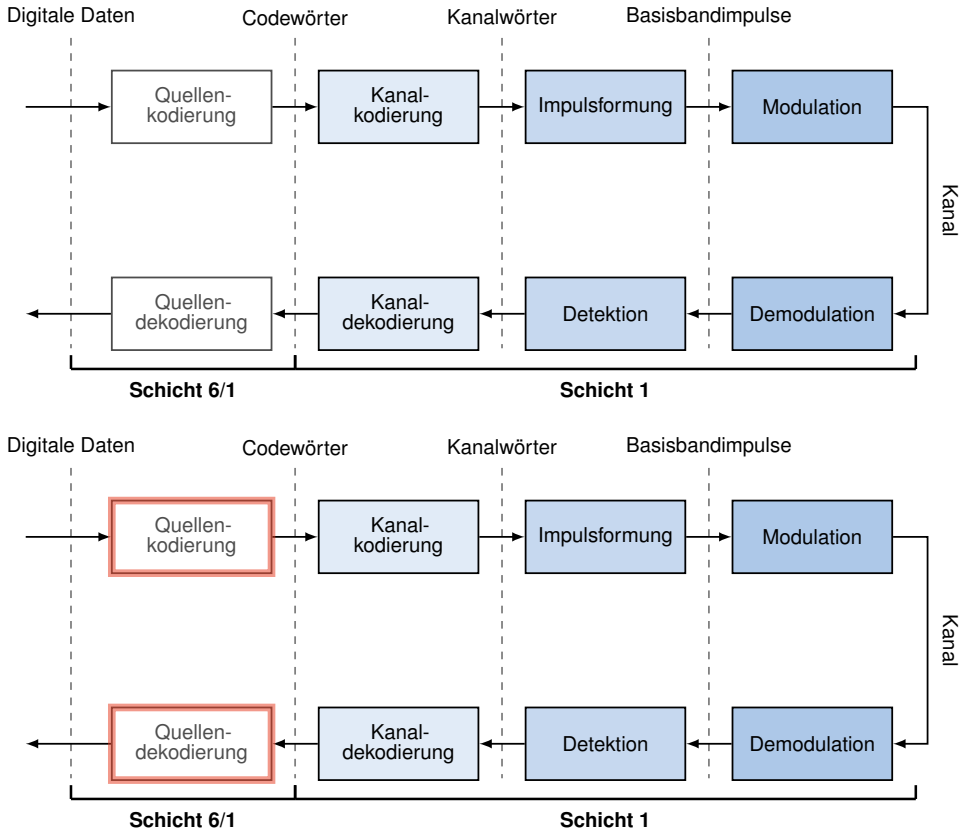
Anmerkungen:

- Das ist nur ein Modell – mit stark vereinfachenden Annahmen.
- Wie man einen Kanalcode mit genau der richtigen Menge Redundanz konstruieren kann, so dass C maximiert wird, ist ein offenes Problem der Informationstheorie. (← Challenge!)
- Wir sprechen hier von Datenraten im informationstheoretischen Sinn, d. h. die zu übertragenden Daten liegen redundanzfrei vor. Dies ist in praktischen Systemen nie gewährleistet:
  - Nutzdaten werden vor dem Senden nicht zwangsläufig (und niemals optimal) komprimiert
  - Zusätzlich zu den Nutzdaten werden Kontrollinformationen (Header) benötigt (→ später)

  ⇒ Die tatsächlich erzielbare Netto-Datenrate liegt unterhalb der informationstheoretischen Schranke.

1.4. Nachrichtenübertragung

1.4.0.2. Nachrichtenübertragung



Quellenkodierung [9]

Quellenkodierung (Source Coding):

Ziel der Quellenkodierung ist es,

- Redundanz aus den zu übertragenden Daten zu entfernen
  durch Abbildung von Bitsequenzen auf Codewörter,
  was einer einer verlustlosen Datenkompression entspricht.

Die Quellenkodierung kann in unterschiedlichen Schichten des ISO/OSI-Modell vorkommen:

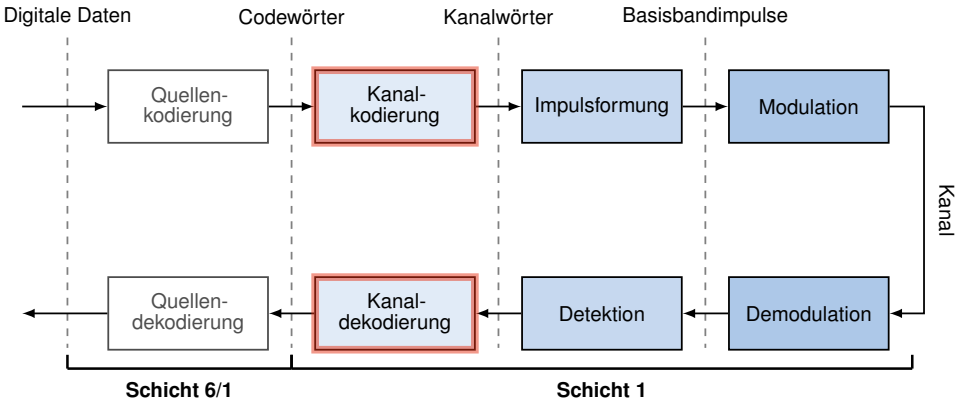
- Datenkompression kann auf der Darstellungsschicht (Schicht 6) stattfinden
- Daten können bereits in komprimierter Form vorliegen (verlustlos komprimierte Dateiformate, z. B. ZIP, PNG, FLAC)
- Im Mobilfunkbereich (digitale Sprachübertragung) kann die Quellenkodierung einer niedrigen Schicht zugeordnet werden
- In lokalen Netzwerken (Ethernet, WLAN) findet i. d. R. keine Quellenkodierung statt

Beispiele:

- Huffman-Code
- Run-Length-Enconding (RLE)

In Kapitel 5 (Darstellungsschicht) gehen wir auf den Huffman-Code ein

1.4.0.3. Übersicht



Kanalkodierung [9] Jeder realisierbare Übertragungskanal ist unzuverlässig. Ein Maßstab dafür ist die Bitfehlerwahrscheinlichkeit p_e:

- Bei Ethernet über Kupferkabel charakteristisch: $p_e \approx 10^{-8}$
- Bei WLAN charakteristisch: $p_e \approx 10^{-6}$ oder mehr
- Bei ungesicherter Funkübertragung charakteristisch: $p_e \approx 10^{-4}$ oder mehr

Gedankenspiel:

- Ungesicherte Funkverbindung mit $p_e = 10^{-4}$, Fehler unabhängig und gleichverteilt
- Paketlänge $L = 1500\,\mathrm{B} = 12000\,\mathrm{bit}$
- $\Pr[\text{„Kein Bitfehler im Paket“}] = (1 - 10^{-4})^{12000} \approx 30\,\%$

$\Rightarrow 70\,\%$ der übertragenen Datenpakete würden mind. einen Bitfehler enthalten. **Kanalkodierung [9]** Jeder realisierbare Übertragungskanal ist unzuverlässig. Ein Maßstab dafür ist die Bitfehlerwahrscheinlichkeit p_e:

- Bei Ethernet über Kupferkabel charakteristisch: $p_e \approx 10^{-8}$

- Bei WLAN charakteristisch: $p_e \approx 10^{-6}$ oder mehr
- Bei ungesicherter Funkübertragung charakteristisch: $p_e \approx 10^{-4}$ oder mehr

Gedankenspiel:

- Ungesicherte Funkverbindung mit $p_e = 10^{-4}$, Fehler unabhängig und gleichverteilt
- Paketlänge $L = 1500\,\mathrm{B} = 12000\,\mathrm{bit}$
- Pr[„Kein Bitfehler im Paket“] $= (1 - 10^{-4})^{12000} \approx 30\,\%$

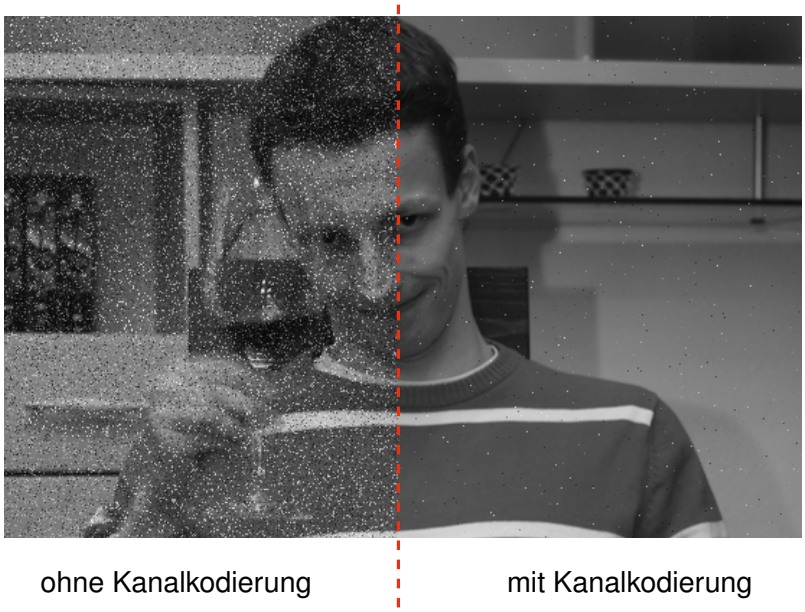
$\Rightarrow$ 70 % der übertragenen Datenpakete würden mind. einen Bitfehler enthalten.

Kanalkodierung (Channel Coding):

Ziel der Kanalkodierung ist es, den zu übertragenden Daten gezielt Redundanz hinzuzufügen, so dass eine möglichst große Anzahl an

- Bitfehlern erkannt und
- korrigiert werden kann.

Können Bitfehler nicht korrigiert, aber erkannt werden, so muss die Fehlermeldung an eine höher liegende Schicht weitergegeben werden, welche dann eine Wiederholung des Datenaustauschs initiiert. Da eine solche Vorgehensweise zeitaufwändiger ist, als eine eigenständige Korrektur ergibt sich hier ein Tradeoff: Je zeitkritischer eine Anwendung und je unzuverlässiger ein Kanal ist, dest erstrebenswerter ist eine Fehlerkorrektur, welche jedoch zu Lasten der Übertragungsrate geht (da mehr Bits für Redundanz aufgewandt werden müssen). Je zuverlässiger ein Kanal ist, desto weniger ist es sinnvoll, die Übertragung für den (damit selteneren) Fehlerfall zu optimieren, wodurch eine höhere Übertragungsrate erzielt wird. **Beispiel:** Unkomprimiertes Bild (Bitmap) über einen verlustbehafteten Kanal versendet



Geringfügige Übertragungsfehler sind in analogen Systemen tolerierbar:

- Rauschen / Knacksen bei einer Telefonverbindung
- „Schnee“ im analogen Fernsehen
- UKW-Radio

In digitalen Systemen können Übertragungsfehler schwerwiegende Konsequenzen haben, z.B.:

- Übertragung komprimierter Daten (ggf. Fehlerfortpflanzung bei der Dekodierung)
- Übertragung verschlüsselter Daten (ggf. Fehlerfortpflanzung bei der Entschlüsselung)
- Fehlerfreie Übertragung kann gefordert sein (z. B. heruntergeladenes Programm kann schon bei einzelnem Bitfehler unbrauchbar sein)

:

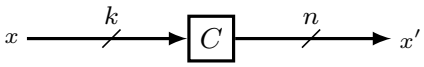
Es werden also zusätzliche Protokolle und Mechanismen benötigt, um trotz Kanalkodierung auftretende Übertragungsfehler

- zumindest zu erkennen und
- bei Bedarf eine Übertragung zu wiederholen.

⇒ Zusammenspiel von Prüfsummen und Quittungsprotokollen,
typischerweise auf Schichten 2, 4 bzw. 7.

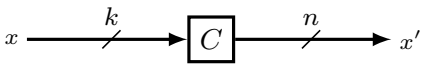
Kanalkodierung: Blockcodes Blockcodes unterteilen den Datenstrom

- in Blöcke der Länge k und
- übersetzen diese in Kanalwörter der Länge $n > k$ wobei
- die zusätzlichen $n - k$ bit für Fehlererkennung und Rekonstruktion verwendet werden.



Das Verhältnis $R = \frac{k}{n}$ wird als Coderate bezeichnet. **Kanalkodierung: Blockcodes** Blockcodes unterteilen den Datenstrom

- in Blöcke der Länge k und
- übersetzen diese in Kanalwörter der Länge $n > k$ wobei
- die zusätzlichen $n - k$ bit für Fehlererkennung und Rekonstruktion verwendet werden.



Das Verhältnis $R = \frac{k}{n}$ wird als Coderate bezeichnet. **Beispiel:** Repetition Code

- $k = 1$, $n = 3$, Abbildung: $0 \mapsto 000$, $1 \mapsto 111$

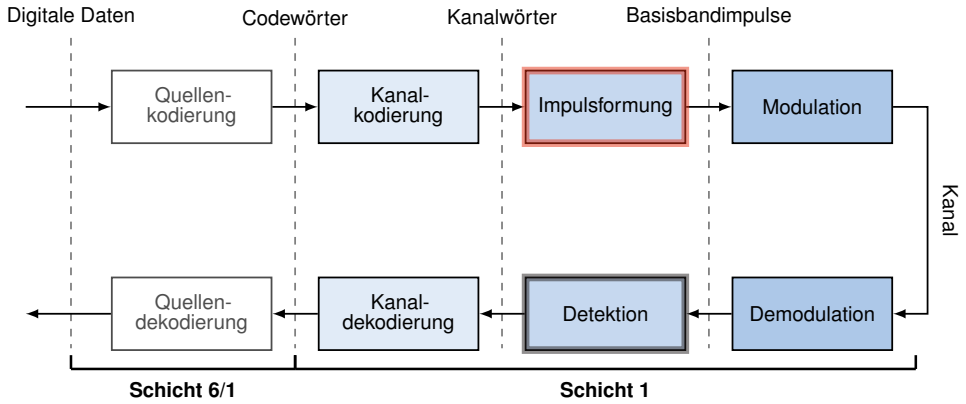
- Dekodierfehler, wenn mind. 2 Bit pro Block verfälscht wurden:

$$\Pr[\text{„Dekodierfehler“}] = \binom{3}{2} p_e^2 (1 - p_e) + \binom{3}{3} p_e^3 \approx \Big|_{p_e = 10^{-4}} 3 \cdot 10^{-8}$$

- Neues Problem:
  - Die zu sendende Anzahl an Bits wird verdreifacht
  - Im fehlerfreien Fall würde die erzielbare Datenrate also auf $R = 1/3$ sinken

  $\Rightarrow$ Kosten-/Nutzenverhältnis zwischen Fehlerwahrscheinlichkeit und Redundanz abhängig von der momentanen Bitfehlerrate

1.4.0.4. Übersicht



Impulsformung [9]

Impulsformung:

Ziel der (Basisband-)Impulsformung ist es, aus einem Datenstrom ein analoges Basisbandsignal (Frequenzen f nahe Null) zu erzeugen. Hierzu werden

- in regelmäßigen Abständen gewichtete Sendegrundimpulse erzeugt,
- von denen jeder ein Bit oder eine Gruppe von Bits darstellt.

Beispiel:

- Gegeben sei der Datenstrom `00 01 10 11 00 11 00 11`
- Rechtecksimpulse werden mit einem von vier Symbolen $d \in \{\pm 0.5, \pm 1.5\}$ gewichtet
- Wir wählen eine Zuordnung zwischen Gruppen von $N = 2$ Bit und den $M = 4$ Symbolen:
  `00` $\mapsto d = -1.5$, `01` $\mapsto d = -0.5$, `10` $\mapsto d = 0.5$, `11` $\mapsto d = 1.5$

Grundimpulse (Beispiele)

- Non-Return-to-Zero-Impuls (NRZ) bzw. Rechtecksimpuls

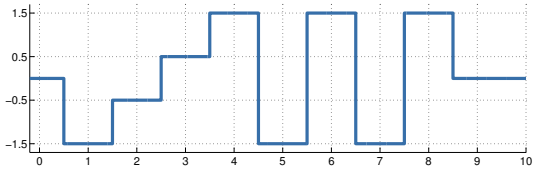
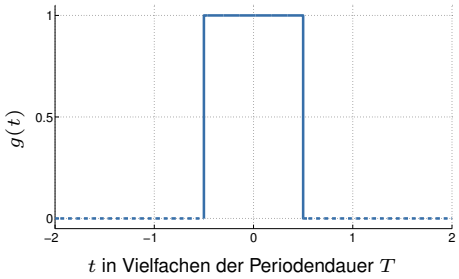
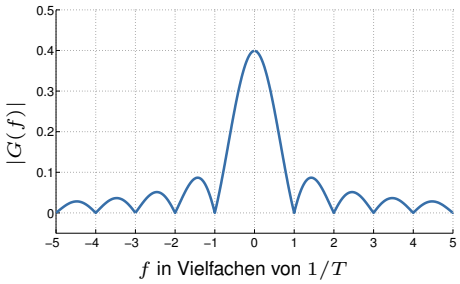
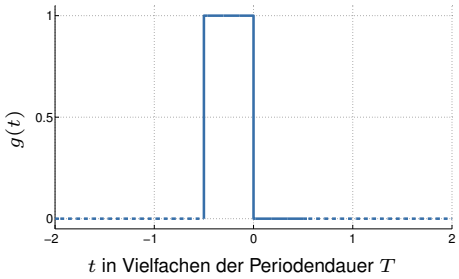


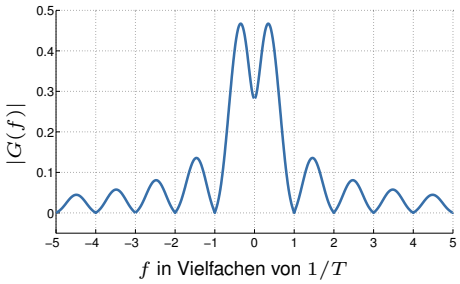
Abbildung 1.10.: Basisbandsignal $s(t) = \sum_{n=1}^{8} d_n \cdot \mathrm{rect}(t - nT)$.





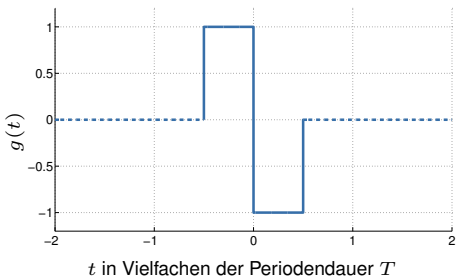
- Return-to-Zero-Impuls (RZ)

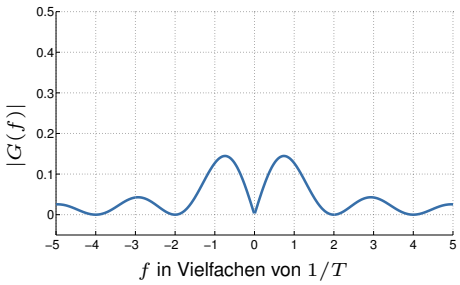




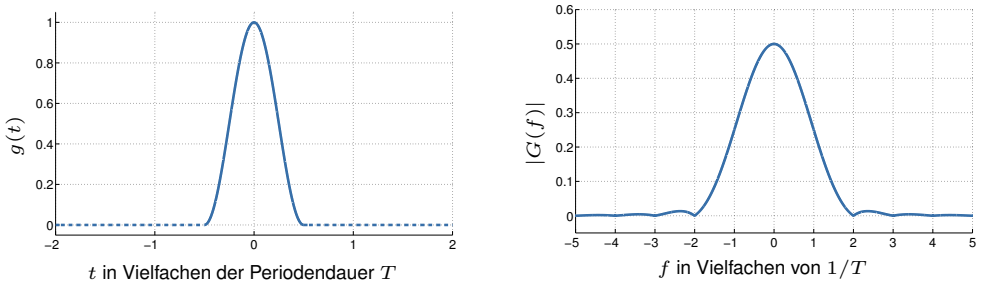
Grundimpulse (Beispiele)

- Manchester-Impuls





- $\cos^2$-Impuls



Leitungscodes Leitungscodes (nicht zu verwechseln mit Kanalcodes) definieren die Abfolge von einer bestimmten Art von Grundimpulsen, welche Bits oder Gruppen von Bits repräsentieren.

Definition: Symbol:

Im Kontext von Leitungscodes verstehen wir unter einem Symbol eine phys. messbare Veränderung des Zeitsignals. Einzelne Grundimpulse bestehen also aus einem oder mehreren Symbolen.

Wichtige Eigenschaften von Leitungscodes:

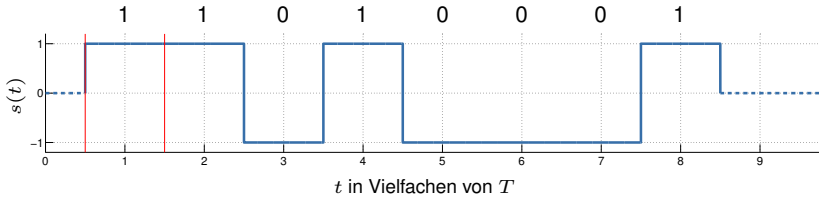
- Anzahl der Signalstufen (binär, ternär, ...)
- Anzahl kodierter Bits pro Symbol
- Schrittgeschwindigkeit (Symbolrate, Baudrate), Einheit bd

Optionale Eigenschaften von Leitungscodes:

- Taktrückgewinnung
- Gleichstromfreiheit
- Bereitstellung von Steuerzeichen (4B5B-Kodierung → später)

Je nach Art der verwendeten Grundimpulse und deren Abfolge haben Leitungscodes Einfluss auf die benötigte Kanalbandbreite. Als Daumenregel gilt: Je mehr abrupte Signalwechsel stattfinden, desto breiter ist das benötigte Spektrum. (s. Beispiele)

Non-Return-To-Zero (NRZ)



Kodiervorschrift:

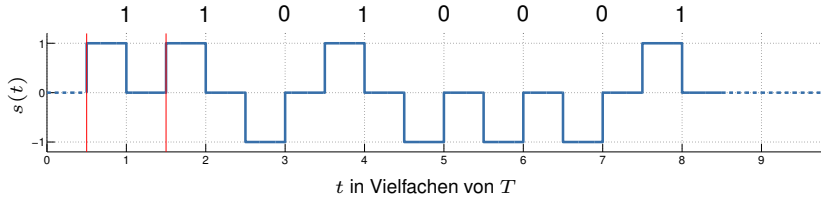
- Grundimpuls $g(t) = \text{rect}(t)$ (Rechtecksimpuls) mit Periodendauer T
- Mögliche Zuweisung der Gewichte $d_n = \begin{cases} 1 & b_n = 1 \\ -1 & b_n = 0 \end{cases}$

- Sendesignal ist definiert als $s(t) = \sum_{n=0}^{\infty} d_n \cdot \text{rect}(t - nT)$

Eigenschaften:

- Binärer Code (lediglich zwei Signalstufen)
- Effizienz 1 Bit/Symbol
- Keine Taktrückgewinnung (lange Null- oder Einsfolgen)
- Keine Gleichstromfreiheit
- Relativ breites Spektrum

Return-To-Zero (RZ)



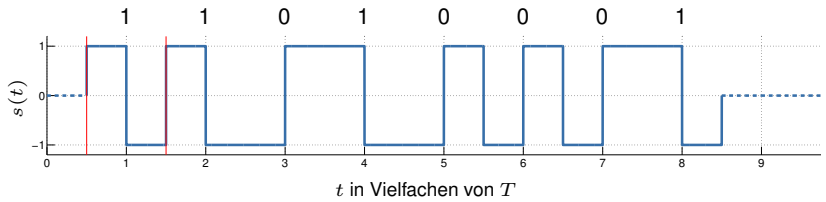
Kodiervorschrift:

- Grundimpuls $g(t) = \text{rz}(t)$ (Return-to-Zero) mit Periodendauer T
- Mögliche Zuweisung der Gewichte $d_n = \begin{cases} 1 & b_n = 1 \\ -1 & b_n = 0 \end{cases}$
- Sendesignal ist definiert als $s(t) = \sum_{n=0}^{\infty} d_n \cdot \text{rz}(t - nT)$

Eigenschaften:

- Binärer Code (lediglich zwei Signalstufen)
- Effizienz 1 Bit/2 Symbole
- Taktrückgewinnung durch erzwungene Pegelwechsel einfach
- Keine Gleichstromfreiheit
- Breiteres Spektrum als NRZ

Manchester-Code



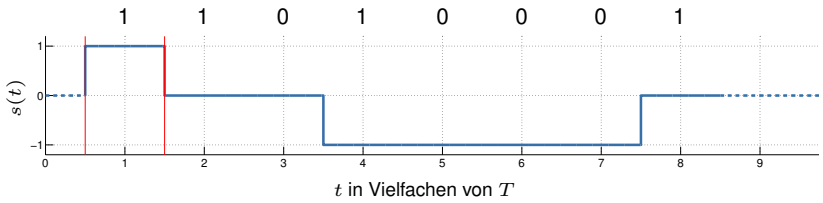
Kodiervorschrift:

- Grundimpuls $g(t) = \text{manc}(t)$ (Manchester) mit Periodendauer T
- Mögliche Zuweisung der Gewichte $d_n = \begin{cases} 1 & b_n = 1 \\ -1 & b_n = 0 \end{cases}$
- Sendesignal ist definiert als $s(t) = \sum_{n=0}^{\infty} d_n \cdot \text{manc}(t - nT)$

Eigenschaften:

- Binärer Code (lediglich zwei Signalstufen)
- Effizienz 1 Bit/2 Symbole
- Taktrückgewinnung durch erzwungene Pegelwechsel einfach
- Gleichstromfreihheit gewährleistet, da jeder Grundimplus für sich gleichstromfrei ist
- Sehr breites und langsam abklingendes Spektrum

Multi-Level-Transmit 3 (MLT3)



Kodiervorschrift:

- Grundimpuls $g(t) = \text{rect}(t)$ (Rechtecksimpuls) mit mit Periodendauer T
- Gewichte $d_n = \sin\left(\frac{\pi}{2}\sum_{k=0}^{n} b_k\right)$ ($\rightarrow$ abhängig von der Anzahl der bislang beobachteten 1-Bits)
- Sendesignal ist definiert als $s(t) = \sum_{n=0}^{\infty} d_n \cdot \text{rect}(t - nT)$

Eigenschaften:

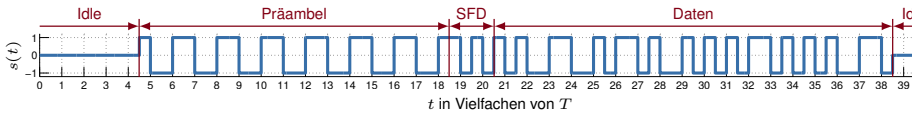
- Tenerär Code (drei Signalstufen)
- Effizienz 1 Bit/Symbol
- Keine Taktrückgewinnung (lange Folge gleicher Bits)
- Keine Gleichstromfreihheit
- Schmales Spektrum, da die Grundperiode durch den periodischen Signalverlauf reduziert wird

Offene Fragen: Wie kann der Empfänger erkennen,

- ob detektierte Symbole überhaupt Daten repräsentieren (Medium könnte „idle“ sein) und
- wie kann der Beginn bzw. das Ende einer Nachricht erkannt werden?

Möglichkeit 1: Coderegelverletzung

- Ist das Medium idle, können ungültige Basisbandimpulse gesendet werden
- Vor Beginn einer Nachricht kann eine fest definierte Anzahl alternierender Bits gesendet werden (Präambel)
- Beginn der Nachricht wird durch eine zweite Sequenz angezeigt (Start Frame Delimiter)
- Dies funktionert mit NRZ, RZ und Manchester Code (z. B. Nullpegel), nicht aber mit MLT3 (Nullpegel bedeutet hier eine Folge von 0-Bits)



Beispiel: Manchester-Code mit Präambel

- Präambel ermöglicht Taktsynchronisation
- Start Frame Delimiter (SFD) am Ende der Präambel signalisiert Beginn der Nachricht
- Coderegelverletzung (Nullpegel) zeigt Idle-Zustand an
- Verwendet bei IEEE 802.3a/i (10 Mbit/s Ethernet über Koaxial- bzw. Twisted-Pair-Kabel → später)

Möglichkeit 2: Steuerzeichen

- Definiere einen Blockcode, welcher Kanalwörter in Gruppen von k Bits unterteilt und auf $n > k$ Bits abbildet
- Dieser Blockcode dient nicht der Fehlerkorrektur (Aufgabe der Kanalkodierung), sondern lediglich der Bereitstellung von Steuerzeichen
- Die Abbildung kann dabei so gewählt sein, dass bei der Übertragung gültiger Kanalwörter
  - Taktrückgewinnung und
  - Gleichstromfreiheit

  auch mit Leitungscodes wie NRZ, RZ und MLT3 möglich werden.
- Ungültige Codewörter, die weder Datenwörter noch Steuerzeichen darstellen, können zur Fehlererkennung verwendet werden

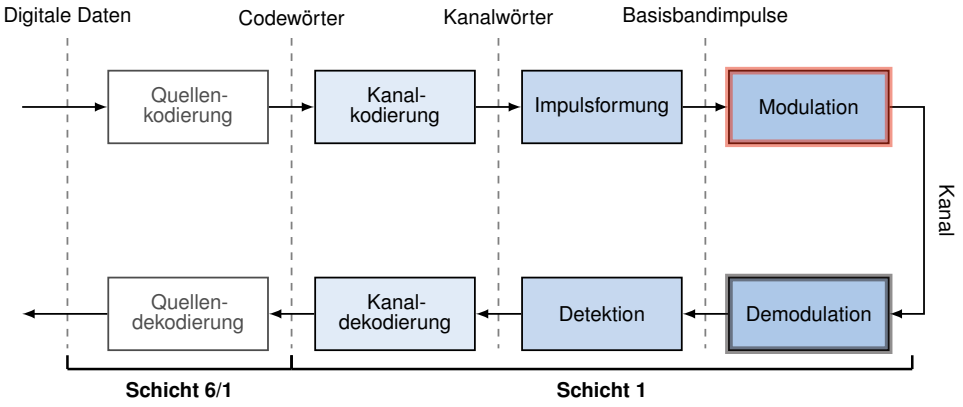
Beispiel 1: 4B5B-Code

- $k = 4$ bit werden auf $n = 5$ bit abgebildet
- Die Zuordnung zwischen Kanalwörtern und Codewörtern wird so gewählt, dass in jedem Block von 5 bit mind. ein Signalwechsel auftritt (Taktrückgewinnung bei NRZ und MLT3)
- Die zusätzlichen Codewörter werden als Steuerzeichen verwendet (Start/Stop, Idle, . . .)
- Verwendet bei IEEE 802.3u (100 Mbit/s FastEthernet über Twisted-Pair-Kabel)

Beispiel 2: 8B10B-Code

- $k = 8$ bit werden auf $n = 10$ bit abgebildet
- Zuordnung ähnlich wie bei 4B5B, allerdings wird hier im zeitlichen Mittel auch Gleichstromfreiheit gewährleistet
- Verwendet u. a. bei PCIe, Serial-ATA, USB . . .

1.4.0.5. Übersicht



Modulation [15] Bislang haben wir nur Basisbandsignale betrachtet:

- Zeitlich verschobene Grundimpulse werden gewichtet
- Zeitlich begrenzte Grundimpulse (wir haben nur solche kennengelernt) besitzen ein unendlich ausgedehntes Spektrum
- Sofern der Übertragungskanal exklusiv für die Basisbandübertragung zur Verfügung steht, ist das zunächst kein Problem

Was ist, wenn der Kanal von mehreren Übertragungen zeitgleich verwendet wird?

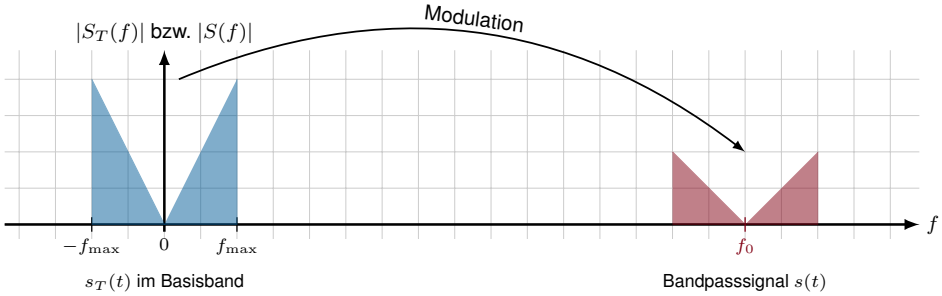
- Das Basisbandsignal (bzw. dessen Grundimpulse) wird tiefpass-gefiltert, was eine Begrenzung des Spektrums (und damit einer leichten Verfälschung des Zeitsignals) entspricht
- Anschließend kann das gefilterte Basisbandsignal auf ein Trägersignal moduliert werden
- Dies entspricht einer Verschiebung des Spektrums (Multiplikation im Zeitbereich ist bedingt eine Verschiebung im Frequenzbereich)
- Teilen sich mehrere Übertragungen auf diese Art einen Kanal, so sprechen wir von Frequency Division Multiplex (FDM)

Prinzipieller Ablauf digitaler Modulationsverfahren

- Die Grundimpulse $g(t)$ werden mittels Tiefpassfilterung auf eine maximale Frequenz $f_{\max}$ beschränkt. Die so gefilterten Impulse bezeichnen wir als $g_T(t)$.
- Das Modulationssignal $s_T(t)$ wird wie im Basisband durch eine gewichtete Überlagerung zeitlich verschobener Grundimpulse erzeugt.
- Das Modulationssignal wird auf ein Trägersignal der Frequenz f_0 aufmoduliert:

$$s(t) = s_T(t) \cdot \cos(2\pi f_0 t) = \left(\sum_{n=0}^{\infty} d_n \cdot g_T(t - nT) \right) \cdot \cos(2\pi f_0 t).$$

Schematischer Ablauf im Frequenzbereich:



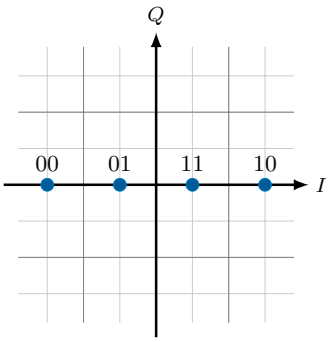
4-ASK (Amplitude Shift Keying)

- Es werden 4 Signalstufen unterschieden $\Rightarrow$ 2 bit/Symbol

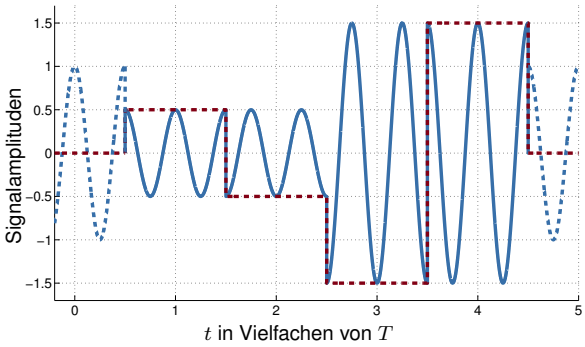
- Es wird nur die Amplitude des Trägersignals moduliert

Beispiel: Signalraum $S = \left\{-\frac{3}{2}, -\frac{1}{2}, \frac{1}{2}, \frac{3}{2}\right\}$

- Je zwei aufeinanderfolgende Bits des Datenstroms werden auf ein Symbol $d \in S$ abgebildet, z. B. $00 \mapsto -\frac{3}{2}$, $01 \mapsto -\frac{1}{2}$, ...
- Die Symbolsequenz d_n verändert die Amplitude eines Grundimpulses (z. B. Rechteckimpuls)
- Das so entstehende Basisbandsignal wird mit einem Trägersignal multipliziert (Modulation)



(a) Signalraumzuordnung



(b) Sendesignal $s(t)$ bzw. unmoduliertes Trägersignal (blau), Modulationssignal $s_T(t)$ (rot)

Quadratur-Amplituden-Modulation (QAM)

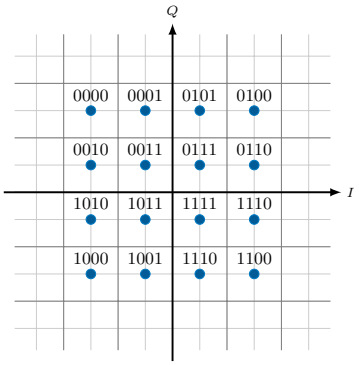
- Man kann kosinus- und sinus-förmige Trägersignale mischen
- Trennung durch Orthogonalität von Sinus und Kosinus möglich
- Der Kosinus wird als Inphase-Anteil, der Sinus als Quadratur-Anteil bezeichnet
- Die Datenrate lässt sich auf diese Weise verdoppeln

$$s(t) = \left(\sum_{n=0}^{\infty} d_{In} \cdot g_T(t - nT)\right) \cos(2\pi f_0 t) - \left(\sum_{n=0}^{\infty} d_{Qn} \cdot g_T(t - nT)\right) \sin(2\pi f_0 t)$$

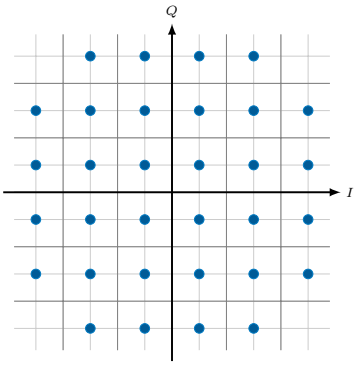
- QAM verdoppelt die Datenrate nochmals?
- Sensationell, wir haben Shannon widerlegt!
- QAM verdoppelt die Datenrate nochmals?
- Sensationell, wir haben Shannon widerlegt!

Natürlich nicht: [16] Durch die Frequenzverschiebung belegt das Bandpasssignal die doppelte Bandbreite im Vergleich zum Basisbandsignal. Es entsteht ein

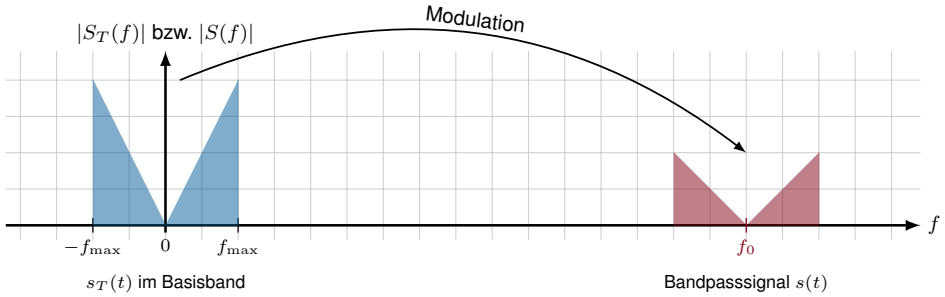
- oberes Seitenband, welches den nicht-negativen Frequenzanteilen im Basisband entspricht, sowie ein
- unteres Seitenband, welches den nicht-positiven Frequenzanteilen im Basisband entspricht.



(a) 16-QAM



(b) 32-QAM



:

- Durch die Modulation wurde also die benötigte Bandbreite verdoppelt
- Dieser „verlorene Freiheitsgrad“ kann durch die Mischung von Sinus- und Kosinus-Trägern wieder kompensiert werden.

Die obere Schranke für die erzielbare Datenrate gilt natürlich weiterhin.

Zusammenfassung Was wir wissen sollten:

- Was sind die Unterschiede und Ziele zwischen Quellenkodierung, Kanalkodierung und Leitungskodierung?
- Wie funktionieren einfache Block-Codes, z. B. Repetition-Code?
- Warum werden trotz aller Kodierverfahren zusätzliche Verfahren zur Fehlererkennung benötigt?
- Wie funktionieren die in diesem Kapitel eingeführten Leitungscodes?
- Was sind die jeweiligen Vor- und Nachteile der hier eingeführten Leitungscodes?
- Wie könnte man diese Leitungscodes auf mehr als zwei oder drei Signalstufen erweitern?
- Was ist das Prinzip von Modulationsverfahren?
- Wie funktioniert Frequenzmultiplex?
- Wie hängen Signalraumzuordnung, Modulationsverfahren und die erzielbare Datenrate zusammen?

- **Für Interessierte:** Wie funktioniert Phase Shift Keying (PSK) und wie sieht eine gültige Signalraumzuordnung für PSK aus?

1.5. Übertragungsmedien

1.5.0.6. Übertragungsmedien

Wir unterscheiden zwischen

- leitungsgebundener und
- nicht-leitungsgebundener Übertragung

sowie zwischen

- akustischen und
- elektromagnetischen Wellen.

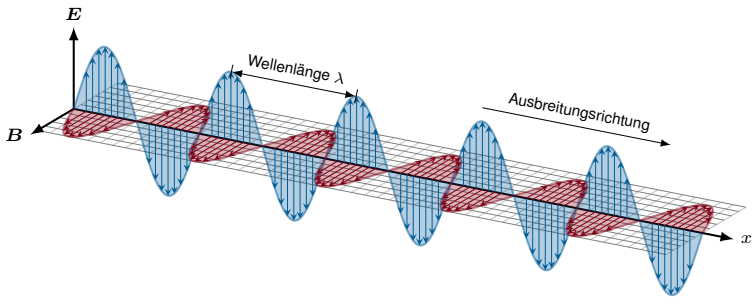
Im Bereich der digitalen Datenübertragung kommen überwiegend elektromagnetische Wellen zum Einsatz. Wenige Ausnahmen bilden hier

- Tonwahlverfahren (z. B. „Einwahl“ bei alten Modemverbindungen) sowie
- einige experimentelle Verfahren, z. B. kabellose Kommunikation unter Wasser.

Im Folgenden verschaffen wir uns einen Überblick über

- Frequenzen im EM-Spektrum,
- was EM-Wellen überhaupt sind und
- und welche Arten von Übertragungsmedien bei leitungsgebundenen Verfahren häufig zum Einsatz kommen.

Elektromagnetische Wellen Elektromagnetische Wellen bestehen aus einer elektrischen ($\boldsymbol{E}$) und magnetischen ($\boldsymbol{B}$) Komponente, welche jeweils orthogonal zueinander und zur Ausbreitungsrichtung stehen:



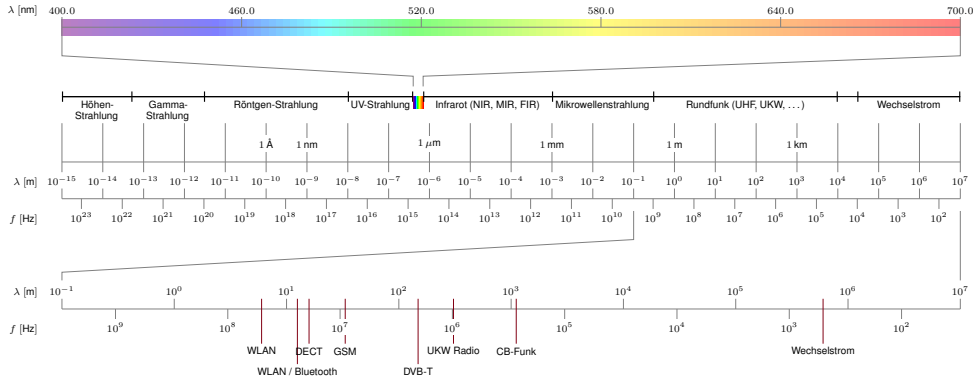
Wichtige Eigenschaften:

- Ausbreitung im Vakuum mit Lichtgeschwindigkeit $c \approx 3 \cdot 10^8\,\mathrm{m/s}$
- Im Gegensatz zu Schallwellen wird kein Medium zur Ausbreitung benötigt

- Innerhalb eines Mediums (Leiter, Luft) beträgt die Ausbreitungsgeschwindigkeit νc, wobei $0 < \nu < 1$ als relative Ausbreitungsgeschwindigkeit bezeichnet wird, z. B. $\nu = 0.9$ in Lichtwellenleitern oder $\nu = 2/3$ in Koaxialleitern
- Die Wellenlänge λ beschreibt die räumliche Ausdehnung einer Wellenperiode
- Die Frequenz f ergibt sich aus Wellenlänge und Lichtgeschwindigkeit zu $f = c/\lambda$

Die Abbremsung der Geschwindigkeit durch das Medium hängt vom Brechungsindex desselben ab.

Elektromagnetisches Spektrum Die untenstehende Abbildung zeigt eine schematische Darstellung des EM-Spektrums:



Zur digitalen Datenübertragung werden überwiegend genutzt:

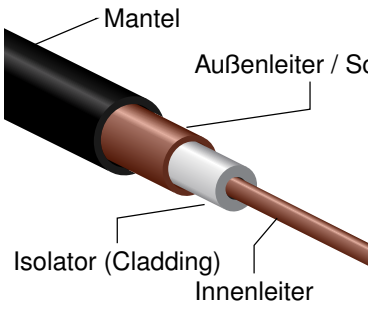
- das Frequenzband zwischen einigen MHz und einigen GHz,
- das optische Spektrum bis zu etwa $\lambda \approx 1\,\text{nm}$ sowie
- Frequenzen im Basisband bis zu einigen hundert MHz.

Koaxialkabel

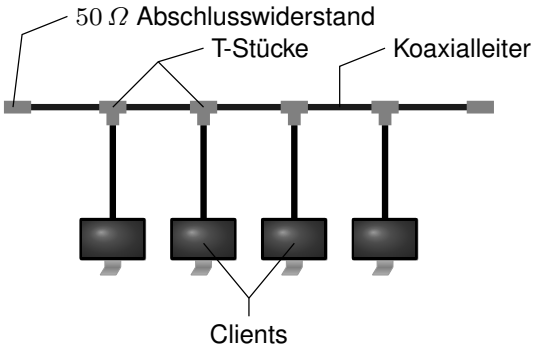
- Eingesetzt u.a. für IEEE 802.3a („10Base2 Ethernet“), 10 Mbit/s
- Ein langer gemeinsamer Bus, an den alle Teilnehmer angeschlossen sind
- Heute (bis auf Kabelverteilnetze, insb. Cable TV, sowie industrielles Umfeld) von geringerer Bedeutung
- Ähnliche Koaxialkabel (mit anderer Dämpfung) kommen im TV-Kabelnetz zum Einsatz

Twisted-Pair-Kabel Allgemeines:

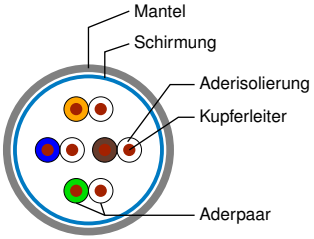
- 2 oder 4 Aderpaare aus Kupferlitzen
- Jedes Aderpaar ist verdrillt (daher die Bezeichnung twisted pair)
- Zweite Ader eines Paares führt inversen Signalpegel (differentielle Kodierung)
- Verdrillung und inverse Signalpegel reduzieren Übersprechen (Crosstalk)
- RJ-45 oder schmalerer RJ-11 Steckverbinder



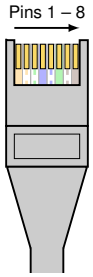
(a) Schematischer Aufbau



(b) 10Base2 Bus (IEEE 802.3a)



(a) Kabelquerschnitt



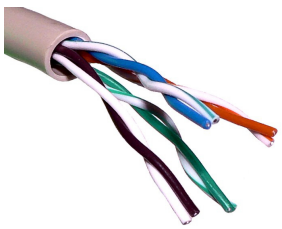
(b) RJ-45 Stecker

Die Verdrillung schützt gegen elektromagnetische Einflüsse von außen. **Verwendung:**

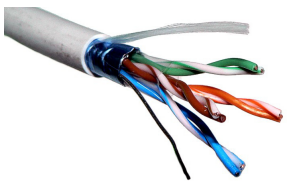
- Lokale Netzwerke (die meisten Ethernet-Standards) mit RJ-45 Steckverbinder
- Telefonanschluss (analog und ISDN) mit RJ-11 Steckverbinder

Je nach Schirmung unterscheidet man

- UTP (unshielded twisted pair)
- STP (shielded twisted pair)
- S/UTP (screened / unshielded twisted pair)
- S/STP (screened / shielded twisted pair)



(a) UTP



(b) Screened UTP

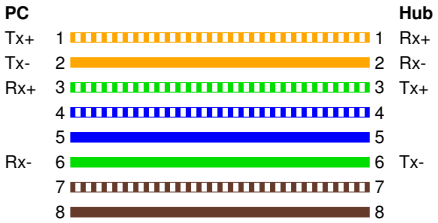
Schirmung hat Einfluss auf

- die Signalqualität (z. B. Übersprechen von elektr. Leitungen) und

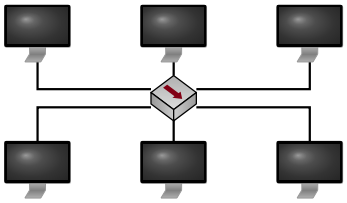
- die Flexibilität der Kabel (gut geschirmte Kabel sind steifer).

Twisted-Pair-Kabel für 100BASE-TX

- Verbindung mehrerer Computer über Hub mittels Straight-Through-Kabel

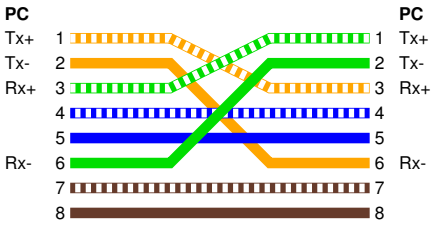


(a) Straight-Through

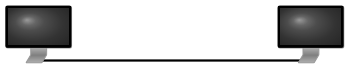


(b) Hub erzeugt phys. Bus, halbduplex

- Direktverbindung zweier Computer mittels Cross-Over-Kabel



(a) Cross-Over



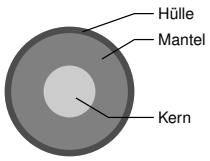
(b) Punkt-zu-Punkt, vollduplex

Optische Leiter

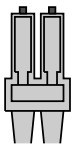
- Licht wird innerhalb des Faserkerns weitergeleitet
- Kern und Mantel besitzen jeweils unterschiedliche optische Dichten
  → Brechungsindex sorgt für annähernde Totalreflexion
- Single-Mode-Fasern vermeiden Streuung durch sehr geringen Kerndurchmesser
  → geringe Verluste, aber sehr empfindlich (Kabelbruch)
- Multi-Mode-Fasern haben einen größeren Kerndurchmesser und neigen daher zum Streuen
  → höhere Verluste, aber weniger empfindlich

Vorteile gegenüber elektrischen Leitern:

- Sehr hohe Datenraten möglich
- Weite Strecken überbrückbar
- Kein Übersprechen
- Galvanische Entkopplung von Sender und Empfänger



(a) Kabelquerschnitt



(b) LC-Stecker (Oberseite)

[[IMAGE DISCARDED DUE TO '/tikz/external/mode=list and make']]

(c) Seitenansicht einer Faser

Zusammenfassung Zur digitalen Kommunikation werden elektromagnetische Wellen

- im Frequenzbereich bis zu einigen GHz bzw.
- im optischen Spektrum genutzt.

Als Übertragungsmedien kommen

- elektrische Leiter (Kupferkabel) sowie
- optische Leiter

in verschiedenen Ausführungen zum Einsatz.

- Funkübertragungen benötigen kein Medium, da sich elektromagnetische Wellen (im Gegensatz zu Schallwellen) im Vakuum ausbreiten
- Das verwendete Medium hat Einfluss auf die Ausbreitungsgeschwindigkeit

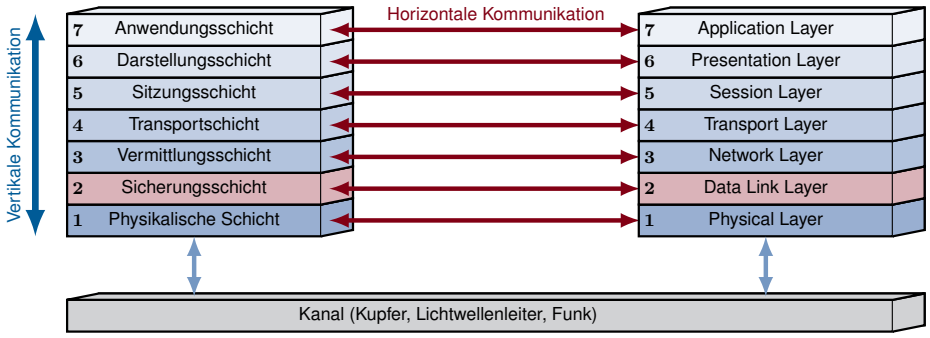
Im nächsten Kapitel beantworten wir die Fragen,

- wie Knoten auf ein ggf. gemeinsames Medium zugreifen können (Medienzugriff) und
- wie Nachrichten an einen bestimmten benachbarten Knoten gesendet werden können (Adressierung).

2. Sicherungsschicht

2.1. Problemstellung und Motivation

2.1.0.7. Einordnung im ISO/OSI-Modell



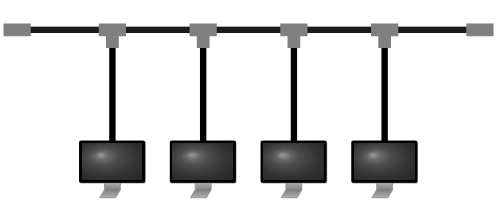
2.1.0.8. Problemstellung und Motivation

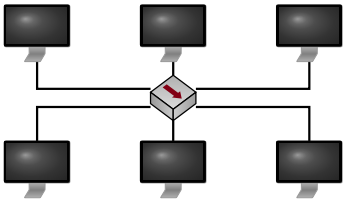
Wir beschäftigen uns zunächst mit sog. Direktverbindungsnetzen, d. h.

- alle angeschlossenen Knoten sind direkt erreichbar und
- werden mittels einfacher Adressen der Schicht 2 identifiziert,
- es findet keine Vermittlung statt,
- eine einfache Weiterleitung (in Form von „Bridging“ oder „Switching“) ist aber möglich.

Beispiele:

- einzelne lokale Netzwerke





- Verbindung zwischen Basisstation und Mobiltelefon
- Bus-Systeme innerhalb eines Computers, z. B. PCIe

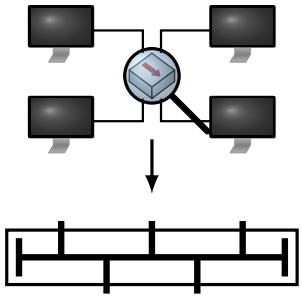
:

Die wesentlichen Aufgaben der Sicherungsschicht sind

- die Steuerung des Medienzugriffs,

Steuerung des Medienzugriffs:

- Hubs z. B. erzeugen nur auf den ersten Blick eine Sterntopologie
- Intern werden alle angeschlossenen Computer zu einem Bus verbunden
- Gleichzeitiges Senden von zwei Stationen führt zu Kollisionen und daher zum Verlust von Nachrichten



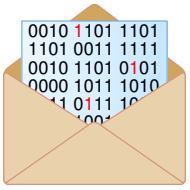
:

Die wesentlichen Aufgaben der Sicherungsschicht sind

- die Steuerung des Medienzugriffs,
- die Prüfung übertragener Nachrichten auf Fehler und

Prüfung übertragener Nachrichten auf Fehler:

- Trotz Kanalkodierung treten Übertragungsfehler auf
- Diese müssen erkannt werden
- Defekte Nachrichten werden nicht an höhere Schichten weitergegeben
- Die **Wiederholung** einer Übertragung ist häufig Aufgabe höherer Schichten



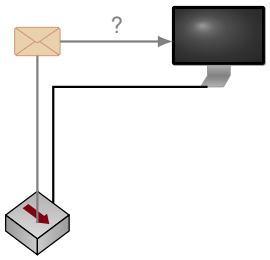
:

Die wesentlichen Aufgaben der Sicherungsschicht sind

- die Steuerung des Medienzugriffs,
- die Prüfung übertragener Nachrichten auf Fehler und
- die Adressierung innerhalb von Direktverbindungsnetzen.

Adressierung:

- Eine Nachricht kann von vielen Knoten empfangen werden, z. B. bei Bus-Verbindungen oder Funknetzwerken
- Der jeweilige Empfänger muss entscheiden können, ob eine Nachricht für ihn bestimmt ist



2.2. Darstellung von Netzwerken als Graphen

2.2.0.9. Darstellung von Netzwerken als Graphen

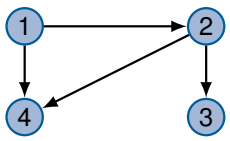
Motivation

- Zur Darstellung von Netztopologien und Knotenverbindungen werden häufig gerichtete oder ungerichtete Graphen verwendet.
- Im Folgenden führen wir die entsprechende Notation und grundlegende Begriffe ein.

Gerichtete Graphen Ein asymmetrisches Netzwerk lässt sich als gerichteter Graph $\mathcal{G} = (\mathcal{N}, \mathcal{A})$ darstellen, wobei

- $\mathcal{N}$ eine Menge von Knoten (Nodes bzw. Vertices) und
- $\mathcal{A} = \{(i, j) \mid i, j \in \mathcal{N} \wedge i, j \text{ sind gerichtet verbunden}\}$ eine Menge gerichteter Kanten (Arcs) bezeichnet.

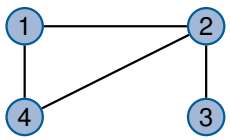
Beispiel: $\mathcal{N} = \{1, 2, 3, 4\}$, $\mathcal{A} = \{(1, 2), (2, 3), (2, 4), (1, 4)\}$



Ungerichtete Graphen Ein symmetrisches Netzwerk lässt sich als ungerichteter Graph $\mathcal{G} = (\mathcal{N}, \mathcal{E})$ darstellen, wobei

- $\mathcal{N}$ eine Menge von Knoten und
- $\mathcal{E} = \{\{i, j\} \mid i, j \in \mathcal{N} \wedge i, j \text{ sind ungerichtet verbunden}\}$ eine Menge ungerichteter Kanten (Edges) bezeichnet.

Beispiel: $\mathcal{N} = \{1, 2, 3, 4\}$, $\mathcal{E} = \{\{1, 2\}, \{2, 3\}, \{2, 4\}, \{1, 4\}\}$



Hinweis zur Notation Ungerichtete Graphen können als gerichtete Graphen mit sym. Kanten verstanden werden. Eine ungerichtete Kante $\{i, j\}$ eines ungerichteten Graphen mit Kantenkosten c_{ij} entspricht also den beiden gerichteten Kanten (i, j) und (j, i) eines gerichteten Graphen mit Kantenkosten $c_{ji} = c_{ij}$.

Pfade in Netzwerken

- Ein Pfad zwischen zwei Knoten[1] $s, t \in \mathcal{N}$ ist eine Menge

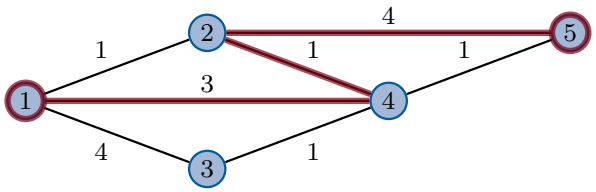
$$\mathcal{P}_{st} = \{(s, i), (i, j), \ldots, (k, l), (l, t)\}$$

  gerichteter Kanten, die s und t miteinander verbinden.

- Die Pfadkosten entsprechen der Summe der Kantenkosten: $c(\mathcal{P}_{st}) = \sum_{(i,j)\in\mathcal{P}_{st}} c_{ij}$.

- Die Pfadlänge entspricht der Anzahl der Kanten auf dem Pfad: $l(\mathcal{P}_{st}) = |\mathcal{P}_{st}|$.
  Die Pfadlänge wird auch Hop Count genannt.

Beispiel: $\mathcal{P}_{15} = \{(1, 4), (4, 2), (2, 5)\}$



$$c(\mathcal{P}_{15}) = 3 + 1 + 4 = 8$$
$$l(\mathcal{P}_{15}) = 3$$

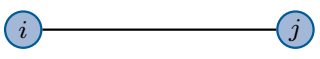
2.2.1. Netztopologien

Netztopologien Die Topologie beschreibt die Struktur, wie Knoten miteinander verbunden sind. Wir unterscheiden die

- physikalische Topologie und die
- logische Topologie.

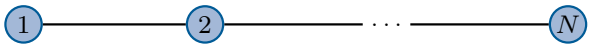
Wichtige Topologien (Beispiele)

- Punkt-zu-Punkt (engl. Point-to-Point)

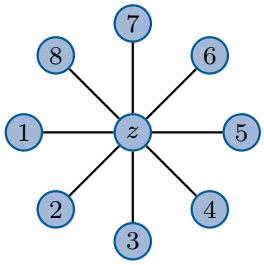


- Kette

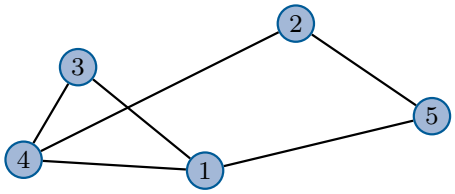
[1] Eine Nachrichtenquelle wird häufig mit s (engl. source) abgekürzt, eine Senke mit t (engl. terminal).



- Stern

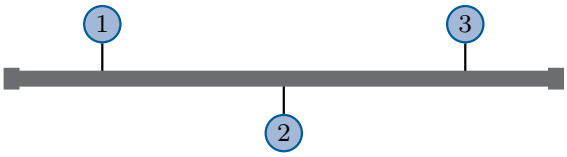


- Vermaschung (engl. Mesh)



- Baum (meist logische Topologie)

- Bus



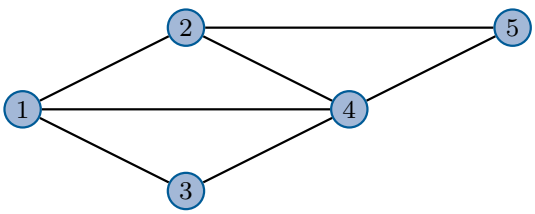
2.2.2. Adjazenz- und Distanzmatrix

Adjazenzmatrix Netzwerke lassen sich leicht als Matrizen schreiben. Die Adjazenzmatrix

$$\boldsymbol{A} = (a)_{ij} = \begin{cases} 1 & \exists (i,j) \in \mathcal{E} \\ 0 & \text{sonst} \end{cases}, \quad \forall i,j \in \mathcal{N}, \qquad \boldsymbol{A} \in \{0,1\}^{N \times N}$$

gibt an, ob Knoten i mit Knoten j verbunden ist. **Beispiel:**

$$\boldsymbol{A} = \begin{bmatrix} 0 & 1 & 1 & 1 & 0 \\ 1 & 0 & 0 & 1 & 1 \\ 1 & 0 & 0 & 1 & 0 \\ 1 & 1 & 1 & 0 & 1 \\ 0 & 1 & 0 & 1 & 0 \end{bmatrix}$$



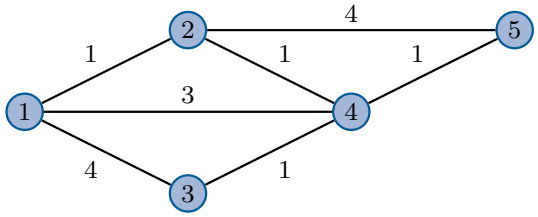
- Das Element a_{ij} der Matrix $\boldsymbol{A}$ ist 1, wenn eine Verbindung von Knoten i zu Knoten j besteht.
- $\boldsymbol{A}$ ist symmetrisch ($\boldsymbol{A} = \boldsymbol{A}^T$), wenn die Kanten ungerichtet sind, d. h. zu jeder Kante (i, j) auch eine antiparallele Kante (j, i) existiert.

Distanzmatrix Die Distanzmatrix

$$\boldsymbol{D} = (d)_{ij} = \begin{cases} c_{ij} & \exists (i,j) \in \mathcal{E} \\ 0 & \text{wenn } i = j\,, \quad \forall i, j \in \mathcal{N}, \\ \infty & \text{sonst} \end{cases} \qquad \boldsymbol{D} \in \mathbb{R}_{0+}^{N \times N}$$

enthält die Kosten der Pfade der Länge 1 zwischen allen Knotenpaaren. **Beispiel:**

$$\boldsymbol{D} = \begin{bmatrix} 0 & 1 & 4 & 3 & \infty \\ 1 & 0 & \infty & 1 & 4 \\ 4 & \infty & 0 & 1 & \infty \\ 3 & 1 & 1 & 0 & 1 \\ \infty & 4 & \infty & 1 & 0 \end{bmatrix}$$



- Das Element d_{ij} der Matrix $\boldsymbol{D}$ gibt die Distanz zwischen Knoten i und Knoten j an.
- Existiert keine direkte Verbindung zwischen i und j, so ist $d_{ij} = \infty$.
- $\boldsymbol{D}$ ist symmetrisch, wenn das Netzwerk symmetrisch ist, d. h. zu jeder Kante (i, j) auch eine antiparallele Kante (j, i) mit denselben Kosten existiert.

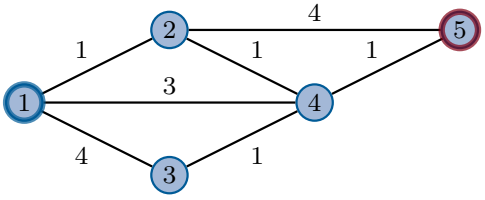
Frage: Wie erhält man die Matrix, welche die Kosten eines kürzesten Pfads zwischen je zwei Knoten enthält? **Antwort:** Man potenziert $\boldsymbol{D}$ bzgl. des min-plus-Produkts

$$\boldsymbol{D}^n = \boldsymbol{D}^{n-1} \otimes \boldsymbol{D} \quad \text{mit} \quad d_{ij}^n = \min_{k \in \mathcal{N}} \left\{ d_{ik}^{n-1} + d_{kj} \right\}.$$

- Die Matrix $\boldsymbol{D}^n$ enthält die Länge eines jeweils kürzesten Pfades über höchstens n Hops.
- Für ein endliches n konvergiert die Potenzreihe, so dass $\boldsymbol{D}^{n+1} = \boldsymbol{D}^n = \boldsymbol{D}^*$.

Beispiel: Wie entsteht Element $(1, 5)$ der Matrix $\boldsymbol{D}^2$?

$$\boldsymbol{D} = \begin{bmatrix} 0 & 1 & 4 & 3 & \infty \\ 1 & 0 & \infty & 1 & 4 \\ 4 & \infty & 0 & 1 & \infty \\ 3 & 1 & 1 & 0 & 1 \\ \infty & 4 & \infty & 1 & 0 \end{bmatrix}$$



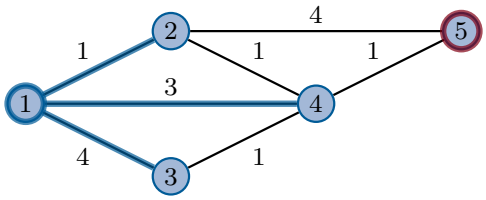
Frage: Wie erhält man die Matrix, welche die Kosten eines kürzesten Pfads zwischen je zwei Knoten enthält? **Antwort:** Man potenziert $\boldsymbol{D}$ bzgl. des min-plus-Produkts

$$\boldsymbol{D}^n = \boldsymbol{D}^{n-1} \otimes \boldsymbol{D} \quad \text{mit} \quad d_{ij}^n = \min_{k \in \mathcal{N}} \left\{ d_{ik}^{n-1} + d_{kj} \right\}.$$

- Die Matrix $\boldsymbol{D}^n$ enthält die Länge eines jeweils kürzesten Pfades über höchstens n Hops.
- Für ein endliches n konvergiert die Potenzreihe, so dass $\boldsymbol{D}^{n+1} = \boldsymbol{D}^n = \boldsymbol{D}^*$.

Beispiel: Wie entsteht Element $(1, 5)$ der Matrix $\boldsymbol{D}^2$?

$$\boldsymbol{D} = \begin{bmatrix} 0 & 1 & 4 & 3 & \infty \\ 1 & 0 & \infty & 1 & 4 \\ 4 & \infty & 0 & 1 & \infty \\ 3 & 1 & 1 & 0 & 1 \\ \infty & 4 & \infty & 1 & 0 \end{bmatrix}$$



- Zeile 1 gibt die Kosten eines jeweils kürzesten Pfades der Länge höchstens 1 von Knoten 1 zu allen anderen Knoten an,

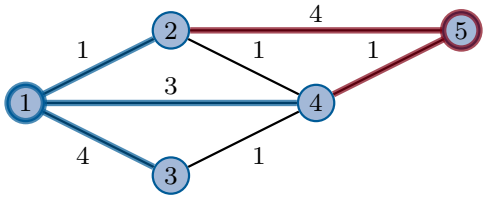
Frage: Wie erhält man die Matrix, welche die Kosten eines kürzesten Pfads zwischen je zwei Knoten enthält? **Antwort:** Man potenziert $\boldsymbol{D}$ bzgl. des min-plus-Produkts

$$\boldsymbol{D}^n = \boldsymbol{D}^{n-1} \otimes \boldsymbol{D} \quad \text{mit} \quad d_{ij}^n = \min_{k \in \mathcal{N}} \left\{ d_{ik}^{n-1} + d_{kj} \right\}.$$

- Die Matrix $\boldsymbol{D}^n$ enthält die Länge eines jeweils kürzesten Pfades über höchstens n Hops.
- Für ein endliches n konvergiert die Potenzreihe, so dass $\boldsymbol{D}^{n+1} = \boldsymbol{D}^n = \boldsymbol{D}^*$.

Beispiel: Wie entsteht Element $(1, 5)$ der Matrix $\boldsymbol{D}^2$?

$$\boldsymbol{D} = \begin{bmatrix} 0 & 1 & 4 & 3 & \infty \\ 1 & 0 & \infty & 1 & 4 \\ 4 & \infty & 0 & 1 & \infty \\ 3 & 1 & 1 & 0 & 1 \\ \infty & 4 & \infty & 1 & 0 \end{bmatrix}$$



- Zeile 1 gibt die Kosten eines jeweils kürzesten Pfades der Länge höchstens 1 von Knoten 1 zu allen anderen Knoten an,
- Spalte 5 gibt die Kosten an, mit denen Knoten 5 von allen anderen Knoten über einen kürzesten Pfad der Länge höchstens 1 erreicht werden kann.

Wie oft muss multipliziert werden?

- Der Wert n, so dass $\boldsymbol{D}^n = \boldsymbol{D}^{n+1} = \boldsymbol{D}^*$ gilt, ist durch den längsten einfachen Pfad im Netzwerk beschränkt.
- Der längste einfache Pfad ist durch die Anzahl N der Knoten beschränkt.

$$\Rightarrow n < N$$

Im vorherigen Beispiel reicht bereits $n = 3$ aus, obwohl $N = 5$ gilt.

:

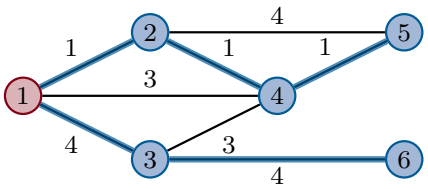
Die Matrix $\boldsymbol{D}^*$ enthält die Kosten eines jeweils kürzesten Pfades zwischen je zwei Knoten und löst damit das All-pair-shortest-distance-Problem (apsd).

2.2.3. Generierung von Baumstrukturen

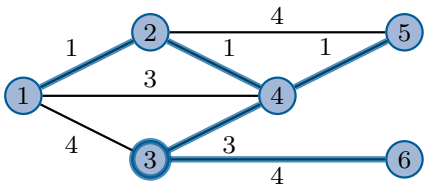
Generierung von Baumstrukturen Ein Baum ist ein zusammenhängender aber schleifenfreier Graph. Wir unterscheiden im folgenden zwei spezielle Arten von Bäumen:

- Shortest Path Tree (SPT)
  Verbindet einen Wurzelknoten mit jeweils minimalen Kosten mit jedem anderen Knoten des Netzwerks.
- Minimum Spanning Tree (MST)
  Verbindet alle Knoten des Netzwerks mit insgesamt minimalen Kosten.

Diese Bäume minimieren unterschiedliche Metriken und sind i. A. **nicht identisch**. **Beispiel:** In Kapitel 3 werden wir zwei Algorithmen zur Erzeugung von SPTs kennen lernen / wiederholen:



(a) Shortest Path Tree (SPT) mit Wurzelknoten 1



(b) Minimum Spanning Tree (MST)

- Algorithmus von Bellman-Ford (basiert auf dem min-plus-Produkt)
- Dijkstras-Algorithmus (Greedy-Prinzip)

2.3. Verbindungscharakterisierung, Mehrfachzugriff, Medienzugriffskontrolle

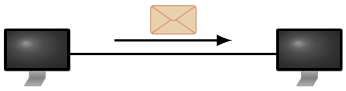
2.3.1. Verbindungscharakterisierung

2.3.1.1. Verbindungscharakterisierung

Eine Verbindung zwischen zwei Knoten kann hinsichtlich einiger grundlegender Eigenschaften charakterisiert werden:

- Übertragungsrate
- Übertragungsverzögerung
- Übertragungsrichtung
- Mehrfachzugriff (Multiplexing)

Zunächst betrachten wir eine Punkt-zu-Punkt-Verbindung:



Übertragungsrate

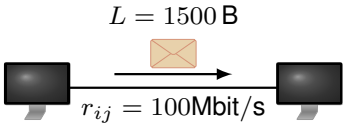
Definition (Übertragungsrate und Serialisierungszeit):

Die Übertragungsrate r in bit/s bestimmt die notwendige Zeit, um L Datenbits auf ein Übertragungsmedium zu legen. Sie bedingt die Serialisierungszeit

$$t_s = \frac{L}{r}.$$

Die Serialisierungszeit bzw. Übertragungsverzögerung wird im Englischen als Serialization Delay bzw. Transmission Delay bezeichnet (vgl. t_s).

Beispiel:



$$t_s = \frac{L}{R} = \frac{1500 \cdot 8\,\text{bit}}{100 \cdot 10^6\,\text{bit/s}} = 120\,\mu\text{s}$$

Frage: Wann empfängt Knoten j das erste Bit der Nachricht?
Ausbreitungsgeschwindigkeit In Kapitel 1 haben wir bereits gesehen, dass Signale i. d. R. elektromagnetische Wellen sind, welche sich mit Lichtgeschwindigkeit im Medium ausbreiten.

Definition (Ausbreitungsverzögerung):

Die Ausbreitungsverzögerung über eine Distanz d rührt von der endlichen Ausbreitungsgeschwindigkeit von Signalen, welche relativ zur Lichtgeschwindigkeit im Vakuum $c \approx 300.000\,\text{km/s}$ angegeben wird:

$$t_p = \frac{d}{\nu c}.$$

Der Wert $0 < \nu < 1$ ist die relative Ausbreitungsgeschwindigkeit in einem Medium. Für Kupfer gilt beispielsweise $\nu \approx 2/3$. Die Ausbreitungsverzögerung wird im Englischen als Propagation Delay bezeichnet (vgl. Benennung t_p).

Beispiel:

- Im Beispiel auf der vorherigen Folie haben wir exemplarisch die Serialisierungszeit zu $t_s = 120\,\mu\text{s}$ bestimmt
- Angenommen die Knoten i und j sind $d_{ij} = 100\,\text{m}$ voneinander entfernt
- Bei Lichtgeschwindigkeit benötigen Signale für diese Strecke gerade einmal 334 ns
  $\Rightarrow j$ empfängt bereits das erste Bit der Nachricht, wenn i gerade das 33ste Bit sendet!

Frage: Wie lange dauert es, bis j das letzte Bit der Nachricht empfangen hat?
Übertragungszeit und Nachrichtenflussdiagramm In einem Nachrichtenflussdiagramm bzw. Weg-Zeit-Diagramm lässt sich die zeitliche Abfolge beim Senden und Empfangen von Nachrichten grafisch veranschaulichen:

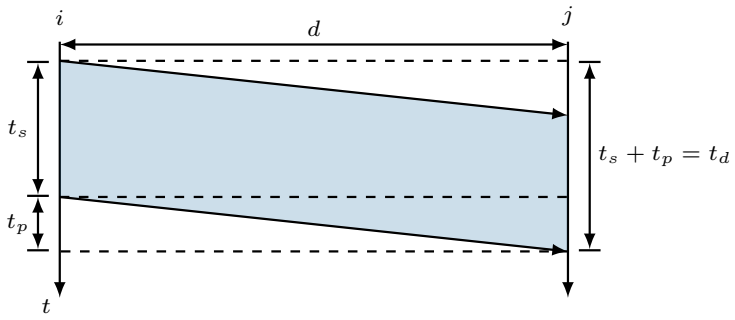


Abbildung 2.1.: Nachrichtenflussdiagramm

- Die Gesamtverzögerung t_d (Delay) ergibt sich daher zu $t_d = t_s + t_p = \frac{L}{r} + \frac{d}{\nu c}$.
- Die Ausbreitungsverzögerung kann bei der Bestimmung von t_d u. U. vernachlässigt werden. Dies hängt allerdings von r, L und d ab! (s. Übung)

Bandbreitenverzögerungsprodukt Durch die endliche Ausbreitungsverzögerung besitzt ein Übertragungskanal eine gewisse „Speicherkapazität“ C, welche als **Bandbreitenverzögerungsprodukt** bekannt ist.

Definition (Bandbreitenverzögerungsprodukt):

Als Bandbreitenverzögerungsprodukt bezeichnet man die Anzahl an Bits (Kapazität)

$$C = t_p r = \frac{d}{\nu c} r,$$

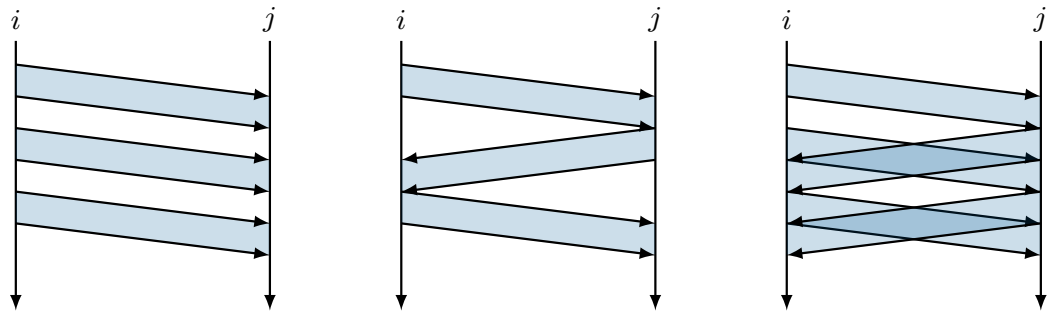
die sich in einer Senderichtung gleichzeitig auf der Leitung befinden können.

Beispiel:

- Leitung mit $r = 1\,\text{GBit/s}$
- Länge $d = 10\,\text{m}$
- $\nu = 2/3$ (Kupferleitung)
- $C = t_p \cdot r = \frac{d}{\nu c} \cdot r = \frac{10\,\text{m}}{2 \cdot 10^8 \frac{\text{m}}{\text{s}}} \cdot 10^9 \frac{\text{Bit}}{\text{s}} \approx 50\,\text{Bit}$

Übertragungsrichtung Hinsichtlich der Übertragungsrichtung unterscheidet man: Die Art der Verbindung hängt dabei ab von

- den Fähigkeiten des Übertragungskanals,
- dem Medienzugriffsverfahren und



(a) Simplex (b) Halbduplex (c) Vollduplex

- den Anforderungen der Kommunikationspartner.

Mehrfachzugriff (Multiplexing) Häufig ist es von Vorteil, Nachrichten unterschiedlicher Teilnehmer gemeinsam über eine Leitung zu übertragen: Ein anderes Zeitmultiplex-Verfahren

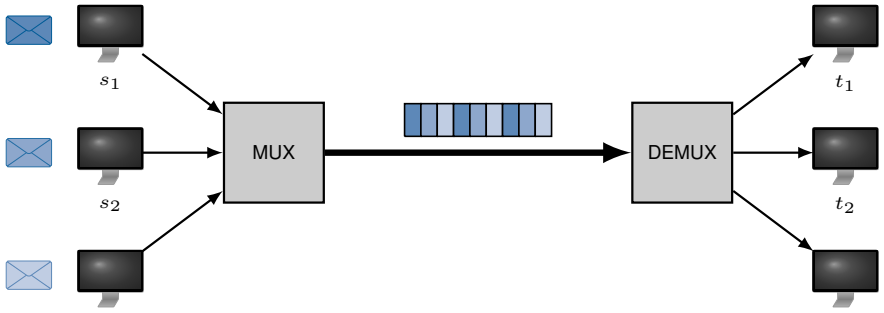


Abbildung 2.2.: Deterministisches Zeitmultiplex-Verfahren

haben wir bereits kennen gelernt:

- Werden mehrere Computer mittels eines Hubs miteinander verbunden,
- so bildet das Hub ein gemeinsames geteiltes Medium,
- auf das die Computer mittels eines nicht-deterministischen Medienzugriffsverfahrens abwechselnd zugreifen.

Übersicht über Multiplex-Verfahren

- **Zeitmultiplex (Time Division Multiplex, TDM)**
  (s. vorherige Folie)
  - Deterministische Verfahren z. B. im Telefonnetz, bei ISDN-Verbindungen und im Mobilfunk
  - Nichtdeterministische Verfahren (konkurrierender Zugriff) in paketbasierten Netzwerken (z. B. Ethernet, WLAN)
- **Frequenzmultiplex (Frequency Division Multiplex, FDM)**
  Aufteilung des Kanals in unterschiedliche Frequenzbänder (spektrale Zerlegung) und Zuweisung Frequenzbänder an Kommunikationspartner (s. Kapitel 1).
  - Omnipräsent bei Funkübertragungen (z. B. unterschiedliche Radiosender)
  - Einsatz bei Glasfaserübertragungen („Modes“ mit unterschiedlicher Farbe)

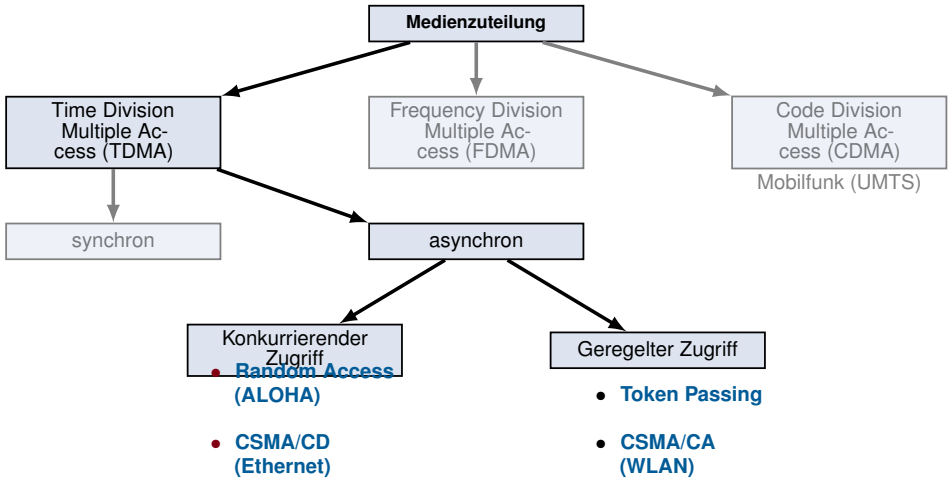
  - Koexistenz von ISDN und DSL auf derselben Leitung

- **Raummultiplex (Space Division Multiplex, SDM)**
  Verwendung mehrerer paralleler Übertragungskanäle.
  - „Kanalbündelung“ bei ISDN
  - MIMO (Multiple-In Multiple-Out) bei kabellosen Übertragungen (Verwendung mehrerer Antennen schafft mehrere Übertragungskanäle)

- **Codemultiplex (Code Division Multiplex, CDM)**
  Verwendung orthogonaler Alphabete und Zuweisung der Alphabete an Kommunikationspartner.
  - Die Mobilfunktechnologie UMTS repräsentiert eine Variante von CDMA
  - Eine weitere Variante von CDMA im Mobilfunkbereich, CDMA2000, findet sich u.a. im Netz des amerikanischen Providers *Verizon* (c.f. „CDMA-iPhone“)

2.3.2. Medienzugriff

Mehrfachzugriff und Medienzugriffskontrolle [10] Einige der (statischen) Multiplexing-Verfahren eignen sich auch als Mehrfachzugriffsverfahren:



Diese ausgewählten vier Zugriffsverfahren werden wir im Folgenden näher kennen lernen.
Bewertungskriterien für Medienzugriffsverfahren sind u.a.

- Durchsatz, d. h. Gesamtanzahl an Nachrichten pro Zeiteinheit, die übertragen werden können
- Verzögerung für einzelne Nachrichten
- Fairness zwischen Teilnehmern, die sich dasselbe Medium teilen
- Implementierungsaufwand für Sender und Empfänger

Problem bei synchronem TDMA

- Der Kanal wird statisch zwischen Teilnehmern aufgeteilt
- Datenverkehr ist aber stossartig bzw. burst-artig, d. h. ein Teilnehmer überträgt kurz mit hoher Bandbreite und danach längere Zeit nicht mehr

- Bandbreite steht während Ruhepausen anderen Teilnehmern nicht zur Verfügung

Lösungsidee: Asynchrones (flexibles) TDMA

- Keine statische Aufteilung / Zuweisung von Zeitslots
- Stattdessen: Zufälliger, konkurrierender oder dynamisch geregelter Medienzugriff

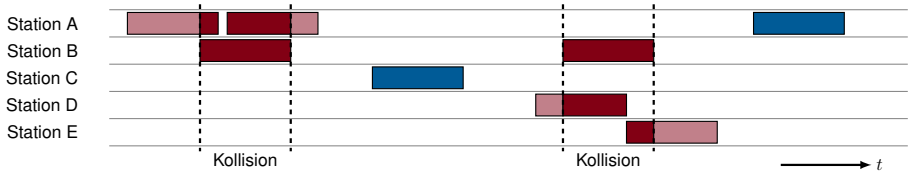
2.3.3. ALOHA und Slotted ALOHA

Random Access (ALOHA)

- Entwickelt an der Universität von Hawaii (1971), c.f. Prof. Abramson
- Ursprünglich für kabellose Datenübertragungen
- Ziel: Verbindung von Oahu mit den anderen hawaiianischen Inseln

Funktionsweise

- Jede Station sendet an eine zentrale Station (vgl. „Basisstation“ in modernen WLANs), sobald Daten vorliegen
- Senden zwei Stationen gleichzeitig, kommt es zu Kollisionen
- Erfolgreich übertragene Nachrichten werden vom Empfänger auf anderer Frequenz quittiert
  („out-of-band“ Bestätigungsverfahren auf Link-Layer, keine Kollisionen zwischen Nachrichten und Bestätigungen)



Das Kanalmodell ist vergleichsweise einfach. Es existieren math. Beschreibungen für den sog. ALOHA Random Access Channel.

Erreichbarer Durchsatz mit ALOHA Vereinfachende Annahmen:

- Mittlere bis beliebig große Anzahl an Knoten ($N > 15$)
- Gleiche, unabhängige und geringe Sendewahrscheinlichkeit auf allen Knoten
- Nachrichten konstanter Größe (Sendedauer T)

Modellierung:

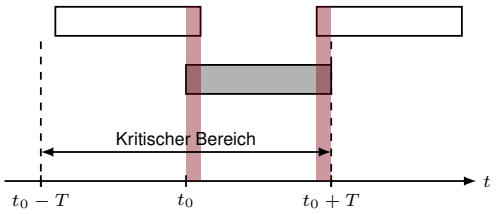
- Ob ein bestimmter Knoten i innerhalb des Zeitintervalls $[t, t+T)$ zu senden beginnt oder nicht entspricht einem Bernoulli-Experiment mit Erfolgs- bzw. Sendewahrscheinlichkeit p_i
- Da die Sendewahrscheinlichkeit für alle Knoten gleich ist, gilt $p_i = p \quad \forall i = 1, \ldots, N$

- Da wir N Knoten haben, die jeweils unabhängig voneinander zu senden beginnen, wird dasselbe Bernoulli-Experiment N-mal wiederholt
- Das ist nichts anderes als eine Binomialverteilung, welche die Anzahl der Erfolge einer Serie gleichartiger und unabhängiger Versuche beschreibt
- Für sinnvoll großes N kann die Binomialverteilung durch eine Poisson-Verteilung[2] approximiert werden (s. Übung)
- Die mittlere erwartete Anzahl von Nachrichten pro Intervall ist gegeben als $Np = \lambda$

Das Ereignis X_t, dass im Intervall $[t, t+T)$ genau k Knoten senden, ist poisson-verteilt:

$$\Pr[X_t = k] = \frac{\lambda^k e^{-\lambda}}{k!}.$$

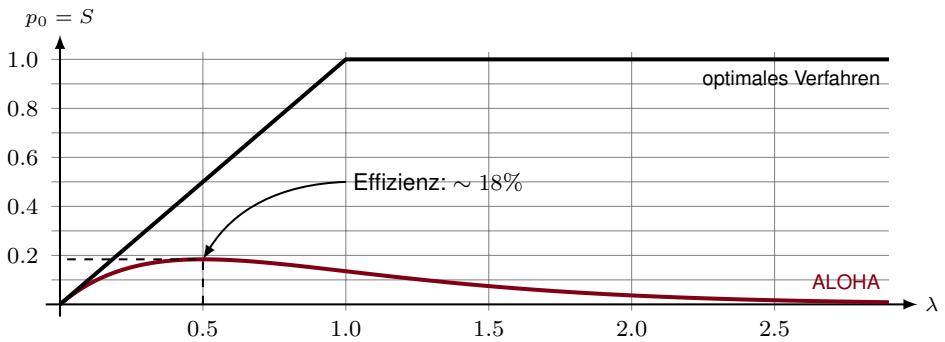
- Eine beliebige Station sende nun zum Zeitpunkt t_0 eine Nachricht
- Eine Kollision tritt genau dann auf, wenn eine andere Station im Intervall $(t_0 - T, t_0 + T]$ versucht, ebenfalls zu übertragen
- Die Übertragung ist also erfolgreich, wenn innerhalb des Intervalls $[t_0, t_0 + T]$ genau eine Übertragung stattfindet und im Intervall $(t_0 - T, t_0)$ keine Übertragung begonnen hat.



- Mit der Dichtefunktion $\Pr[X_t = k] = \frac{\lambda^k e^{-\lambda}}{k!}$ erhalten wir
- die Wahrscheinlichkeit p_0 für eine erfolgreiche Übertragung:

$$p_0 = \Pr[X_{t_0-T} = 0] \cdot \Pr[X_{t_0} = 1] = \mathrm{e}^{-\lambda} \cdot \lambda \mathrm{e}^{-\lambda} = \lambda \mathrm{e}^{-2\lambda}$$

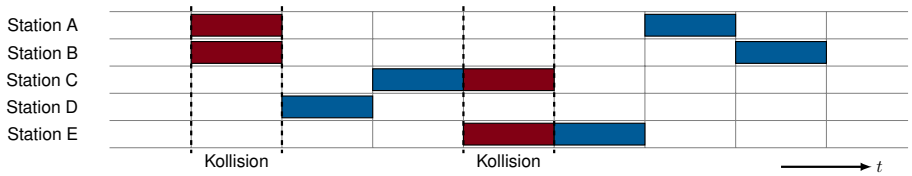
Die Erfolgswahrscheinlichkeit p_0 kann gegen die Senderate λ aufgetragen werden:



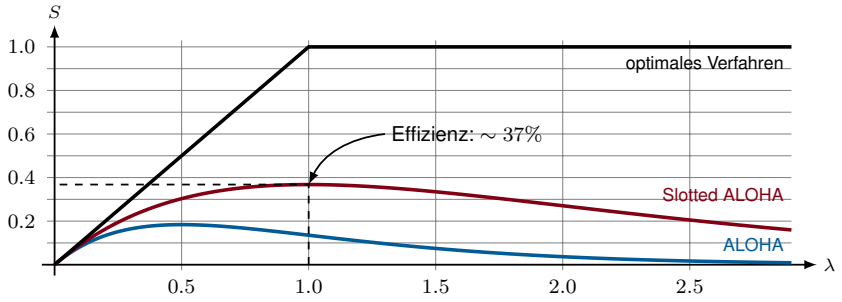
[2] Verteilung der seltenen Ereignisse

- Wir wissen, dass innerhalb eines beliebigen Intervalls $[t, t+T)$ höchstens eine Übertragung erfolgreich sein kann
- Dementsprechend entspricht die Anzahl S der erfolgreichen Nachrichten pro Intervall gleichzeitig der Wahrscheinlichkeit für eine erfolgreiche Übertragung
- Bei einem optimalen Verfahren würde die Anzahl erfolgreicher Nachrichten S linear mit der Senderate ansteigen, bis die maximale Anzahl von Nachrichten pro Zeitintervall erreicht ist (hier ist das genau eine Nachricht pro Intervall)
- Steigt die Senderate weiter, würde dies ein optimales Verfahren nicht beeinträchtigen

Variante: Slotted ALOHA Stationen dürfen nicht mehr zu beliebigen Zeitpunkten mit einer Übertragung beginnen, sondern nur noch zu den Zeitpunkten $t = nT$, $n = 0, 1, \ldots$



Kritischer Bereich ist nur noch T anstelle von $2T \Rightarrow S = \lambda \cdot e^{-\lambda}$.



2.3.4. CSMA, CSMA/CD, CSMA/CA

Carrier Sense Multiple Access (CSMA) Eine einfache Verbesserung von Slotted ALOHA: „Listen Before Talk“

- Höre das Medium ab
- Beginne erst dann zu senden, wenn das Medium frei ist

Non-persistent CSMA:

1. Wenn Medium frei, übertrage im nächstmöglichen Intervall
2. Wenn belegt, warte eine feste Zeitspanne, dann (1)

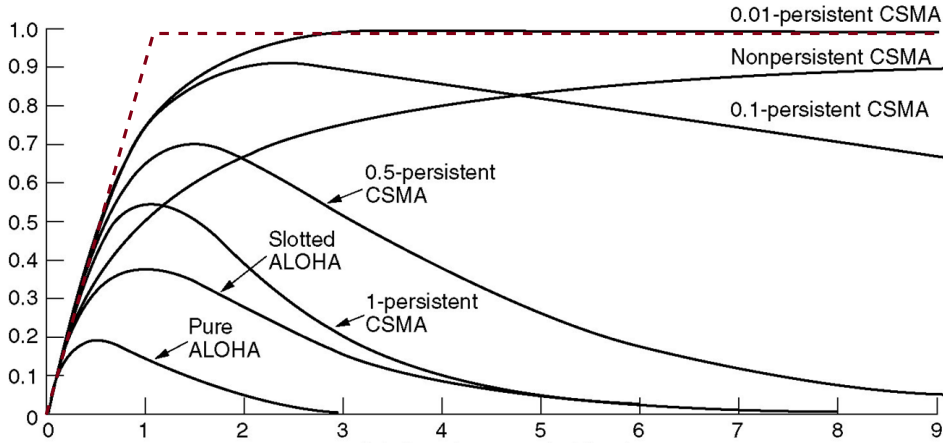
1-persistent CSMA:

1. Wenn Medium frei, übertrage im nächstmöglichen Intervall
2. Wenn Medium belegt, warte bis frei und übertrage im nächstmöglichen Intervall

p-persistent CSMA:

1. Wenn Medium frei, übertrage mit Wahrscheinlichkeit p oder verzögere mit Wahrscheinlichkeit $1 - p$ um eine Slotzeit
2. Wenn Medium belegt, warte bis frei, dann (1)
3. Wenn um eine Slotzeit verzögert, dann (1)

Alle bisherigen Verfahren im Vergleich



CSMA/CD (Collision Detection) Ansatz

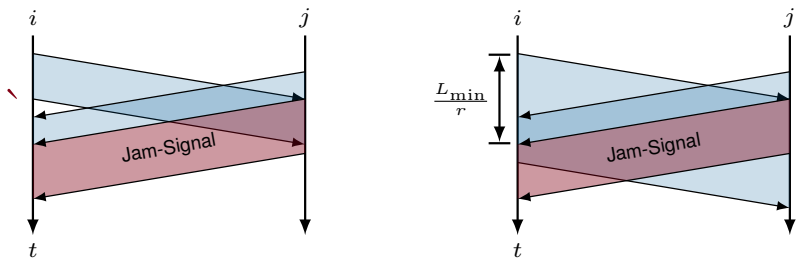
- Erkenne Kollisionen und wiederhole die Übertragung, wenn eine Kollision erkannt wird
- Verzichte auf das Senden von Bestätigungen
- Wird keine Kollision erkannt, gilt die Übertragung als erfolgreich

Problem: Der Sender muss die Kollision erkennen, während er noch überträgt

Voraussetzung für CSMA/CD [10]:

Angenommen zwei Stationen i und j kommunizieren über eine Distanz d mittels CMSA/CD. Damit Kollisionen erkannt werden können, müssen Nachrichten folgende Mindestlänge $L_{\min}$ aufweisen:

$$L_{\min} = \frac{2d}{\nu c} r$$



Wird 1-persistentes CSMA mit Kollisionserkennung verwendet, ergibt sich folgendes Problem:

- Die Kollision zerstört die Nachrichten beider in die Kollision verwickelten Stationen
- Mind. eine der Stationen sendet ein JAM-Signal
- Nachdem das Medium frei wird, wiederholen beide Stationen die Übertragung
  $\Rightarrow$ Es kommt sofort wieder zu einer Kollision

Lösung: Warte „zufällige“ Zeit nach einer Kollision

Binary Exponential Backoff:

Bei der k-ten Wiederholung einer Nachricht

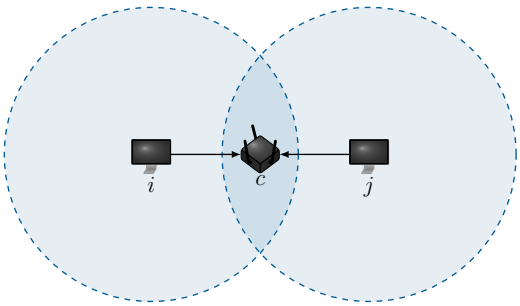
- wählt der Sender zufällig $n \in \{0, 1, \ldots, \min\{2^{k-1}, 1024\}\}$ aus und
- wartet n Slotzeiten vor einem erneuten Sendeversuch.

Die maximale Wiederholungszahl beträgt $k = 15$ (also 16 Sendeversuche).

Durch die Wartezeiten, die

- zufällig gewählt und
- situationsabhängig größer werden,
- wird die Kollisionswahrscheinlichkeit bei Wiederholungen reduziert.

CSMA/CA (Collision Avoidance) In Funknetzwerken funktioniert CSMA/CD nicht, da der Sender einer Nachricht eine Kollision auch bei ausreichender Nachrichtenlänge nicht immer detektieren kann. **„Hidden Station“:**



- Knoten i und j senden gleichzeitig
- Knoten c erkennt die Kollision
- Weder i noch j bemerken die Kollision

CSMA/CA basiert auf p-persistentem CSMA, d. h.

1. Wenn Medium frei, übertrage mit Wahrscheinlichkeit p oder verzögere mit Wahrscheinlichkeit $1 - p$ um eine Slotzeit
2. Wenn Medium belegt, warte bis frei, dann (1)
3. Wenn um eine Slotzeit verzögert, dann (1)

Fallbeispiel: IEEE 802.11 DCF (Distributed Coordination Function)

- Festes Zeitinervall zwischen Rahmen (DCF Interframe Spacing).
- Wenn Medium mind. für DIFS idle ist, dann wähle unabhängig und gleichverteilt eine Anzahl von Backoff-Slots aus dem Intervall $\{0, 1, 2, \ldots, \min\{2^{c+n} - 1, 255\}\}$.
- c ist abhängig vom PHY (z.B. $c = 4$), n ist der Retry Counter des Binary Exponential Backoffs.

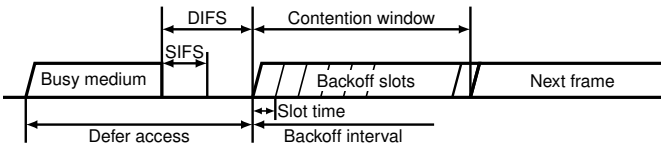


Abbildung 2.3.: IEEE 802.11 DCF

- Medienzugriff hat durch festes c stets ein Contention Window.
- Ein Rahmen gilt in IEEE 802.11 als erfolgreich übertragen, wenn
  - im Fall von Unicasts der Empfänger eine Bestätigung schickt (Link-Layer Acknowledgements) oder
  - im Fall von Broadcasts die Übertragung eines Frames störungsfrei abgeschlossen wird.
- Da i.d.R. nicht gleichzeitig gesendet und das Medium geprüft werden kann (anders bei Ethernet), ist die zweite Bedingung praktisch bereits erfüllt, wenn ein Knoten zu senden beginnt.

Fallbeispiel: IEEE 802.11 DCF (Distributed Coordination Function) Was passiert in der Praxis? Beispiel anhand handelsüblicher Hardware im Monitor Mode[3]:

- Wir deaktivieren Link-Layer Bestätigungen und prüfen, wie sich die Hardware verhält.
- Ohne Bestätigungen wird es keinen Exponential Back-Off geben, da Übertragungen (einmal begonnen) nicht mehr fehlschlagen (IEEE 802.11 macht kein Media Sensing während eine Übertragung läuft).
- Das Contention Window sollte aber $\{0, 1, 2, \ldots, 15\}$ betragen und Backoff-Slots unabhängig und gleichverteilt daraus gezogen werden. $\Rightarrow$ Ein zu sendendes Frame sollte (bei freiem Medium) im Mittel um 7.5 Slotzeiten verzögert werden.

Was wir nun genau tun:

- Wir können die Verzögerung zwischen aufeinander folgenden Frames einer Station mittels einer zweiten Station relativ genau messen.
- Aus den gemessenen Zeiten erstellen wir eine (empirische) kummulative Verteilungsfunktion (KVF).

[3] Monitor Mode bezeichnet einen Operationsmodus von IEEE 802.11 Hardware, in dem die Netzwerkkarten alle eingehenden Frames vollständig unverarbeitet zugänglich machen, unabhängig davon, ob es sich um Daten-, Management- oder Control-Frames handelt und unabhängig davon, ob das Frame überhaupt an die jeweilige Station adressiert war. Umgekehrt können in diesem Modus auch beliebige Link-Layer Frames „von Hand gebaut“ und unverändert verschickt werden.

- Diese EKVF gibt $\Pr[X \leq N]$ an, also die Wahrscheinlichkeit, dass die Anzahl der gewarteten Slotzeiten (die Größe des Contention Windows) kleiner oder gleich N Slotzeiten ist.
- Diese sollte einer Treppenfunktion zwischen 0 und 15 mit äquidistanten Inkrementen um jeweils 1 Slotzeit folgen.

Fallbeispiel: IEEE 802.11 DCF (Distributed Coordination Function)

[[Image Discarded Due To '/tikz/external/mode=list and make']]

Abbildung 2.4.: EKVF der zeitl. Abstände zwischen Frames einer Station gemessen in Vielfachen von Slotzeiten.

Fallbeispiel: IEEE 802.11 DCF (Distributed Coordination Function) Was bedeutet das nun?

- Während eine dieser Broadcom-Karten sendet, haben alle anderen Sendepause.
- Hält sich ein Gerät nicht an die Vorgaben, kann es sich beim Senden auf Kosten anderer „Vorteile“ verschaffen, da kleinere Contention Phases
  - die Wahrscheinlichkeit erhöhen, die Contention Phase zu gewinnen und
  - natürlich die Idle-Time des Medium reduzieren.
- Gerade Letzteres (Idle-Times) ist bei IEEE 802.11 der limitierende Faktor, da die Zeit zwischen Frames im Verhältnis zur Serialisierungszeit sehr hoch ist.

Anmerkungen:

- Das Beispiel diskutiert IEEE 802.11 Hardware im Monitor Mode und ist daher für den (praxisrelevanten) Infrastructure Mode nur wenig aussagekräftig.
- In einer Masterarbeit haben wir aber kürzlich gezeigt, dass es dort aber nicht viel besser aussieht...
- Das Verhalten hängt nicht nur von der Hardware/Firmware, sondern auch vom Treiber ab.
- Nein, wir werden nicht von Atheros/Qualcomm bezahlt. :)
- Das dazu passende Paper gibts in Moodle.

Erweiterung: RTS/CTS (Request to Send / Clear to Send)

- Übertragungen werden i.d.R. von einer Basisstation gesteuert
- Bevor ein Knoten eine Nachricht überträgt, wird nach dem CSMA/CA-Prinzip ein RTS an die Basisstation geschickt
- Nur wenn die Basisstation mit einem CTS antwortet, darf die Übertragung beginnen

Beispiel:

1. A sendet RTS, welches von B aufgrund der Distanz nicht empfangen wird.

2. Basistation antwortet mit CTS, welches von A und B empfangen wird.

3. A darf senden, B muss eine im CTS definierte Zeitspanne abwarten, bevor überhaupt ein RTS gesendet werden darf.

Erweiterung: RTS/CTS (Request to Send / Clear to Send) Vorteile:

- Kollisionen mit Hidden Stations werden vermieden, aber nicht gänzlich verhindert.
- Insgesamt weniger Kollisionen, auch ohne Hidden Stations.

Nachteile:

- Es können noch immer Kollisionen auftreten, z.B. wenn B das CTS nicht empfängt.
- RTS/CTS nimmt vorab Zeit in Anspruch, was die maximal erzielbare Datenrate reduziert.

Anmerkungen:

- RTS/CTS ist Bestandteil des sog. Virtual Carrier Sensing, da mit dem CTS das Medium für eine bestimmte Zeitspanne für eine Übertragung reserviert wird.
- Um die Verlustwahrscheinlichkeit von RTS/CTS-Nachrichten zu minimieren, werden diese mit der robustesten Kodierung übertragen, was i.d.R. der niedrigsten unterstützten Datenrate entspricht. Im Gegenzug sind RTS/CTS-Nachrichten sehr klein.
- Es ist streng genommen für RTS/CTS nicht notwendig, dass ein Netzwerk durch eine Basisstation kontrolliert wird. Es funktioniert auch im ad-hoc Modus[4] oder (mit Einschränkungen) in Mesh-Netzwerken.
- Alle Geräte, unabhängig davon ob sie zum selben Service Set[5] gehören oder nicht, sollten CTS-Nachrichten verarbeiten.

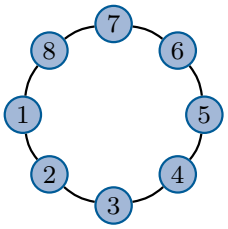
2.3.5. Token Passing

Token Passing Idee: Kollisionsfreie Übertragung durch Weitergabe eines **Tokens**

- Stationen werden zu einem physikalischen Ring zusammengeschaltet
- Ein Token zirkuliert im Ring

[4] Bezeichnet eine Gruppe IEEE 802.11-fähiger Geräte, welche ohne Basistation direkt miteinander kommunizieren
[5] Bezeichnung für eine Gruppe miteinander kommunizierender IEEE 802.11-fähiger Geräte

- Will eine Station senden, nimmt sie das Token vom Ring und darf danach als einzige Station im Ring übertragen
- Nachdem alle Nachrichten gesendet wurden (oder nach einer definierten Zeitspanne), wird das Token wieder auf den Ring gelegt



Empfang von Nachrichten:

- Die Nachricht zirkuliert wie das Token durch den Ring
- Der Empfänger markiert die Nachricht als gelesen und schickt sie weiter
- Trifft sie wieder beim Sender ein, so nimmt dieser sie vom Netz

Was ist, wenn das Token „verloren geht“?

- Es gibt eine Monitor-Station, z. B. die Station, die das erste Token erzeugt hat
- Diese Monitor-Station erzeugt bei Bedarf neue Tokens, entfernt endlos kreisende Pakete und entfernt doppelte Token
- Fällt die Monitor-Station aus, wird von den verbleibenden Stationen eine Neue gewählt

Vorteile:

- Sehr effizient, da keine kollisionsbedingten Wiederholungen
- Garantierte maximale Verzögerung (Determinismus)

Nachteile bzw. Schwierigkeiten:

- Geht das Token verloren, muss es durch ein Neues ersetzt werden
  → eine Station muss spezielle Aufgaben übernehmen (Monitor-Station)
- Fehlerhaftes verhalten eines Knotens stört die gesamte Kommunikation im Ring
- Übertragungsverzögerung u. U. größer als bei CSMA, da Sender auf Token warten muss
- Zusammenschaltung der Stationen zu einem Ring ist u. U. aufwendig

Einsatz heute:

- Token Ring (IEEE 802.5) wurde vollständig von Ethernet (IEEE 802.3) ersetzt und spielt in lokalen Netzwerken heute keine Rolle mehr.
- FDDI (Fiber Distributed Data Interface) ist ein Sammelbegriff für Glasfaserringe bis zu einer Länge von einigen hundert Kilomtern. Diese werden z. B. als Backbone lokaler Zugangsanbieter im städtischen Maßstab eingesetzt.

Zusammenfassung In diesem Teilkapitel haben wir einige flexible Zeitmultiplexverfahren kennengelernt, die Zugriff mehrerer Stationen auf ein gemeinsames Medium erlauben. Im Gegensatz zu statischem Zeitmultiplex wird die Kanalbandbreite nicht für inaktive Knoten reserviert. **Konkurrierender Zugriff:**

- ALOHA und Slotted ALOHA
- CSMA (non-persistent, 1-persistent, p-persistent)
- CSMA/CD (Kollisionserkennung)
  IEEE 802.3 Ethernet

Geregelter Zugriff:

- CSMA/CA (Kollisionsvermeidung)
  IEEE 802.11 WLAN
- Token Passing (Kollisionsverhinderung)
  IEEE 802.5 Token Ring, Fiber Distributed Data Interface (FDDI)

2.4. Rahmenbildung, Adressierung und Fehlererkennung

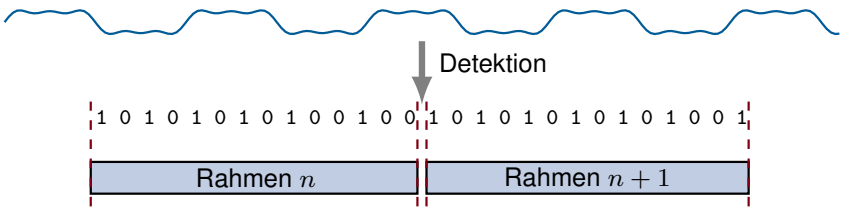
Motivation Bislang haben wir nur von Nachrichten gesprochen, ohne uns Gedanken über deren Format zu machen. Aus Sicht der physikalischen Schicht ist eine Nachricht lediglich eine Folge von Bits. Für eine Betrachtung der Sicherungsschicht reicht diese Vorstellung aber nicht mehr aus. Im Folgenden wollen wir uns Gedanken machen,

- wie einzelne Nachrichten auseinandergehalten werden können,
- welche zusätzlichen Informationen Protokolle der Sicherungsschicht benötigen und
- wie Übertragungsfehler, die trotz Kanalkodierung auftreten, erkannt werden können.

Im Kontext der Sicherungsschicht bezeichnen wir Nachrichten fortan als Rahmen (engl. Frame).

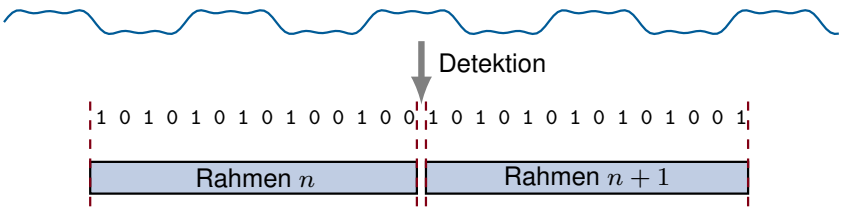
2.4.1. Erkennung von Rahmengrenzen und Codetransparenz

2.4.1.1. Erkennung von Rahmengrenzen



Wie kann der Empfänger Rahmen erkennen, insbesondere wenn

- Rahmen unterschiedliche Größen haben und
- nicht ständig Nutzdaten auf der Leitung liegen (Idle-Perioden)?



Wie kann der Empfänger Rahmen erkennen, insbesondere wenn

- Rahmen unterschiedliche Größen haben und
- nicht ständig Nutzdaten auf der Leitung liegen (Idle-Perioden)?

Es gibt viele Möglichkeiten:

- Längenangabe der Nutzdaten
- Steuerzeichen (Start / Ende)
- Begrenzungsfelder und „Bit-Stopfen“
- Coderegelverletzung

:

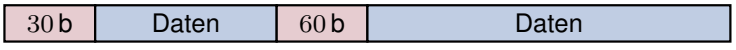
Ziel aller Verfahren zur Rahmenbegrenzung ist die Erhaltung der **Codetransparenz**, d. h. die Übertragung beliebiger Zeichenfolgen zu ermöglichen.

Längenangabe der Nutzdaten Idee:

- Am Anfang des Rahmens steht die Länge der nachfolgenden Nutzdaten (oder die Gesamtlänge des Rahmens).
- Voraussetzung: Das Längenfeld und damit der Beginn einer Nachricht muss eindeutig zu erkennen sein

Längenangabe der Nutzdaten Idee:

- Am Anfang des Rahmens steht die Länge der nachfolgenden Nutzdaten (oder die Gesamtlänge des Rahmens).
- Voraussetzung: Das Längenfeld und damit der Beginn einer Nachricht muss eindeutig zu erkennen sein



Wie kann der Beginn eines Rahmens erkannt werden?

- Durch Steuerzeichen (Start / Ende)
- Durch Voranstellen von Begrenzungsfeldern
- Durch Verlust des Trägersignals zwischen den Rahmen (Coderegelverletzung)

Steuerzeichen In Kapitel 1 haben wir bereits den 4B5B-Code kennengelernt, welcher in Kombination mit Leitungscodes wie MLT-3 auf der physikalischen Schicht eingesetzt wird.

- Je 4 bit Eingabe werden auf 5 bit Ausgabe abgebildet
- Einem Rahmen werden die Startsymbole J/K vorangestellt
- Nach einem Rahmen werden die Endsymbole T/R eingefügt

Eingabe	Ausgabe	Bedeutung	Eingabe	Ausgabe	Bedeutung
0000	11110	Hex data 0	-	00000	Quiet (Signalverlust)
0001	01001	Hex data 1	-	11111	Idle (Pause)
0010	10100	Hex data 2	-	11000	Start #1 (J)
0011	10101	Hex data 3	-	10001	Start #2 (K)
0100	01010	Hex data 4	-	01101	End (T)
0101	01011	Hex data 5	-	00111	Reset (R)
⋮	⋮	⋮	-	11001	Set
1111	11101	Hex data F	-	00100	Halt

Beispiel:
Eingabe: `1011 0101 0110`
Ausgabe: `11000 10001 10111 01011 01110 01101 00111`

Steuerzeichen werden nicht nur auf Schicht 1/2 verwendet. Auf Schicht 6 (Darstellungsschicht) wird der **ASCII-Code (American Standard Code for Information Interchange)** verwendet (7 bit Codeworte):

ASCII (hex)	Zeichen	ASCII (hex)	Zeichen	ASCII (hex)	Zeichen	ASCII (hex)	Zeichen
00	NUL	20	SP	40	@	60	‘
01	SOH	21	!	41	A	61	a
02	STX	22	„	42	B	62	b
03	ETX	23	#	43	C	63	c
04	EOT	24	$	44	D	64	d
05	ENQ	25	%	45	E	65	e
06	ACK	26	&	46	F	66	f
07	BEL	27	’	47	G	67	g
08	BS	28	(	48	H	68	h
09	TAB	29	)	49	I	69	i
0A	LF	2A	*	4A	J	6A	j
0B	VT	2B	+	4B	K	6B	k
0C	FF	2C	,	4C	L	6C	l
0D	CR	2D	-	4D	M	6D	m
0E	SO	2E	.	4E	N	6E	n
0F	SI	2F	/	4F	O	6F	o
10	DLE	10	0	50	P	70	p
11	DC1	11	1	51	Q	71	q
12	DC2	12	2	52	R	72	r
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮

Was ist, wenn Steuerzeichen zufällig in den Nutzdaten vorkommen?

1. Im Fall des 4B5B-Code kann das nicht passieren:
   - 4 bit Datenworte werden injektiv auf 5 bit Datenworte abgebildet
   - Einige der verbleibenden 5 bit Worte werden als Steuerzeichen verwendet

Was ist, wenn Steuerzeichen zufällig in den Nutzdaten vorkommen?

1. Im Fall des 4B5B-Code kann das nicht passieren:
   - 4 bit Datenworte werden injektiv auf 5 bit Datenworte abgebildet

- Einige der verbleibenden 5 bit Worte werden als Steuerzeichen verwendet

2. Der ASCII-Code ist lediglich eine Interpretationsvorschrift:
   - Einige Codeworte sind Textzeichen (Ziffern, Zahlen, ...), andere Steuerzeichen
   - Um ein Steuerzeichen als Datum übertragen zu können, wird dieses durch ein spezielles Steuerzeichen markiert: Escape Character
   - Soll dieses spezielle Steuerzeichen selbst übertragen werden, so wird es verdoppelt
   - Dieses Vorgehen bezeichnet man als **Character Stuffing**

Beispiele für Character Stuffing: Meist wird automatisch für **Codetransparenz** (d.h. Übertragung beliebiger Sequenzen von Zeichen) gesorgt, so dass sich der Benutzer nicht darum kümmern muss. Das trifft nicht auf Porgrammiersprachen zu:

```
System.out.println("Ein \" muss escaped werden");
```

Innerhalb des auszugebenden Strings müssen Anführungszeichen mittels eine Backslashs escaped werden. Weitere Beispiele:

- Bash (Control + C)
- Telnet-Verbindungen
- Texteditoren (Emacs)

Begrenzungsfelder und Bit-Stopfen Idee:

- Markiere Start und Ende einer Nachricht mit einer bestimmten Bitfolge
- Stelle sicher, dass die Markierung nicht zufällig in den Nutzdaten vorkommt (Bit-Stopfen, engl. Bit Stuffing)

Begrenzungsfelder und Bit-Stopfen Idee:

- Markiere Start und Ende einer Nachricht mit einer bestimmten Bitfolge
- Stelle sicher, dass die Markierung nicht zufällig in den Nutzdaten vorkommt (Bit-Stopfen, engl. Bit Stuffing)

Beispiel:

- Start- / Endemarkierung sei `01111110`
- Um das Auftreten der Markierung in Nutzdaten zu verhindern, füge nach fünf aufeinanderfolgenden 1-en eine 0 ein

Eingabe:		1100101111110111111	
Ausgabe:	01111110	11001011111**0**101111**1**0**1**	01111110

- Empfänger entfernt nach fünf aufeinanderfolgenden 1-en die darauf folgende 0

Coderegelverletzung Viele Leitungscodes (z. B. RZ und Manchester) besitzen unabhängig von den zu übertragenden Daten bestimmte Signalwechsel. **Idee:**

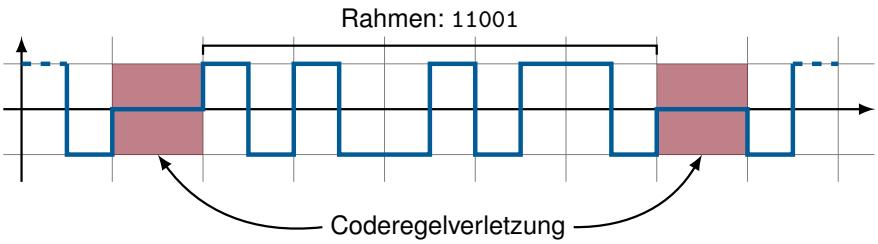
- Lasse bestimmte Signalwechsel aus

- Auf diese Art wird ein ungültiges (im Code nicht existierendes) Symbol erzeugt
- Dieses kann verwendet werden, um Start und Ende von Rahmen zu markieren

Coderegelverletzung Viele Leitungscodes (z. B. RZ und Manchester) besitzen unabhängig von den zu übertragenden Daten bestimmte Signalwechsel. **Idee:**

- Lasse bestimmte Signalwechsel aus
- Auf diese Art wird ein ungültiges (im Code nicht existierendes) Symbol erzeugt
- Dieses kann verwendet werden, um Start und Ende von Rahmen zu markieren

Beispiel: Manchester-Code



Fallbeispiele IEEE 802.3a/i (Ethernet): 10 Mbit/s

- Als Leitungscode wird der Manchester-Code verwendet.
- Das Ende eines Frames wird durch Coderegelverletzung angezeigt.

IEEE 802.3u (FastEthernet): 100 Mbit/s

- Als Leitungscode wird MLT-3 in Kombination mit dem 4B5B-Code verwendet.
- Start und Ende von Rahmen werden durch Steuerzeichen des 4B5B-Codes markiert.

Zusätzlich wird bei IEEE 802.3a/i/u jedem Rahmen noch eine **Präambel** vorangestellt. Diese dient allerdings nur der Taktsynchronisierung zwischen Sender und Empfänger.

2.4.2. Adressierung und Fehlererkennung

2.4.2.1. Adressierung und Fehlererkennung

Bislang wissen wir,

- wie ein binärer Datenstrom übertragen wird und
- wie der Empfänger Rahmengrenzen wiedererkennt.

Wir wissen aber noch nicht,

- wie Nutzdaten, die von Schicht 3 und höher kommen, von der Sicherungsschicht behandelt werden,
- wie der Empfänger eines Rahmens adressiert wird und
- wie aus den Nutzdaten und protokollspezifischen Informationen ein Rahmen entsteht.

Adressierung In Direktverbindungsnetzen

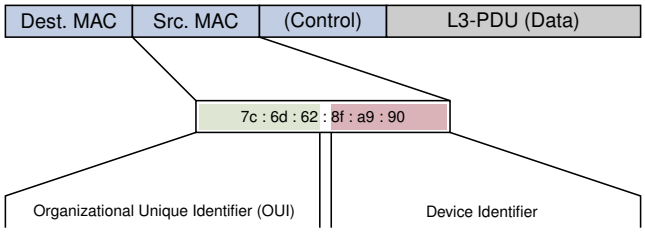
- sind angeschlossene Knoten direkt erreichbar,
- es findet also keine Vermittlung (engl. Routing) zwischen Knoten statt.

Anforderungen an Adressen auf Schicht 2:

- **Eindeutige Identifikation** eines Knotens im Direktverbindungsnetz
- Eine **Broadcast-Adresse**, mittels der **alle** Knoten im Direktverbindungsnetz adressiert werden können.

Adressen auf Schicht 2 bezeichnet man allgemein als **MAC-Adressen**, wobei MAC für Media Access Control steht. **Beispiel:**

MAC-Adressen haben häufig den folgenden Aufbau:

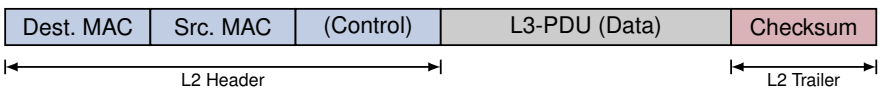


- Netzwerkkarten besitzen ab Werk eine MAC-Adresse. Diese ist meist im ROM (Read Only Memory) der Netzwerkkarte hinterlegt.
- Die Auftrennung in OUI und Device ID ermöglicht es den Herstellern von Netzwerkkarten, eindeutige MAC-Adressen vergeben.
- Daher ist möglich, den Hersteller einer Netzwerkkarte anhand deren MAC-Adresse zu identifizieren (z. B. 7c:6d:62 $\hat{=}$ Apple).
- Als Broadcast-Adresse ist ff:ff:ff:ff:ff:ff („all ones“) definiert.

Fehlererkennung

- Trotz Kanalkodierung können Übertragungsfehler (Bitfehler) auftreten.
- Es kann daher passieren, dass eine fehlerhafte Payload an höhere Schichten weitergeleitet wird.

Um die Wahrscheinlichkeit für derartige Fehler zu minimieren, werden **fehlererkennende Codes** eingesetzt (sog. **Prüfsummen**):



:

Im Gegensatz zur Kanalkodierung dient die Prüfsumme eines Schicht-2-Protokolls üblicherweise nicht der Fehlerkorrektur sondern lediglich der Fehlererkennung.

Cyclic Redundancy Check (CRC) [11]

- CRC berechnet zu einem gegebenen Datenblock (z. B. L3-PDU) eine Checksumme fester Länge.
- Deren Länge wird durch den Grad eines **Generatorpolynoms** $g(x)$ bestimmt.
- Dessen Koeffizienten stammen aus dem finiten Feld $\mathrm{GF}(2) = \{0, 1\}$.
- Die Berechnung der Checksumme ist eine Polynomdivision über $\mathrm{GF}(2)$, weswegen Addition und Subtraktion sich auf bitweise XOR-Operationen reduzieren.

Ethernet verwendet CRC-32:

$$g(x) = x^{32} + x^{26} + x^{23} + x^{22} + x^{16} + x^{12} + x^{11} + x^{10} + x^8 + x^7 + x^5 + x^4 + x^2 + x + 1$$

Beispiel: Generator $g(x) = x^3 + x^2 + 1$, Daten $m(x) = x^7 + x^5 + x^2 + 1$

1. Koeffizienten bestimmen: $g(x) \mathrel{\hat{=}} 1101$ und $m(x) \mathrel{\hat{=}} 10100101$

Beispiel: Generator $g(x) = x^3 + x^2 + 1$, Daten $m(x) = x^7 + x^5 + x^2 + 1$

1. Koeffizienten bestimmen: $g(x) \mathrel{\hat{=}} 1101$ und $m(x) \mathrel{\hat{=}} 10100101$

2. $\mathrm{grad}(g(x)) = 3 \Rightarrow$ Daten mit x^3 multiplizieren. Dies entspricht dem „Anhängen“ von 3 Nullen: $m'(x) = m(x) \cdot x^3 \mathrel{\hat{=}} 10100101000$

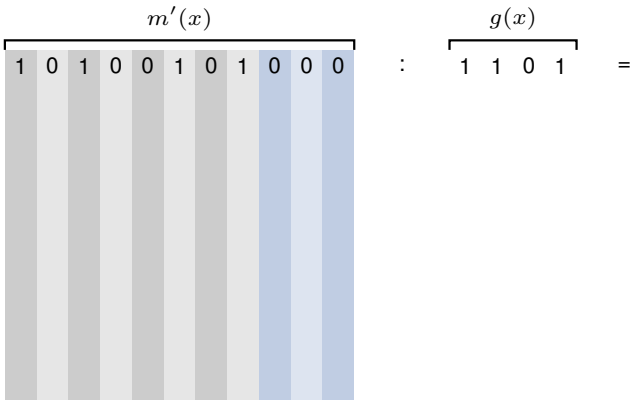
Beispiel: Generator $g(x) = x^3 + x^2 + 1$, Daten $m(x) = x^7 + x^5 + x^2 + 1$

1. Koeffizienten bestimmen: $g(x) \mathrel{\hat{=}} 1101$ und $m(x) \mathrel{\hat{=}} 10100101$

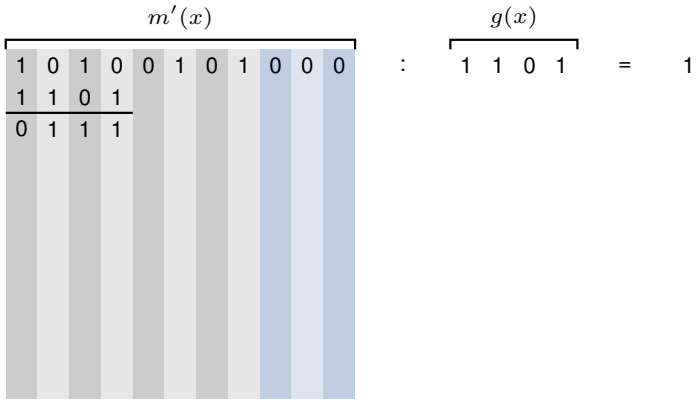
2. $\mathrm{grad}(g(x)) = 3 \Rightarrow$ Daten mit x^3 multiplizieren. Dies entspricht dem „Anhängen“ von 3 Nullen: $m'(x) = m(x) \cdot x^3 \mathrel{\hat{=}} 10100101000$

3. Polynomdivision $m'(x)/g(x)$ ausführen und den Rest $r(x)$ bestimmen.

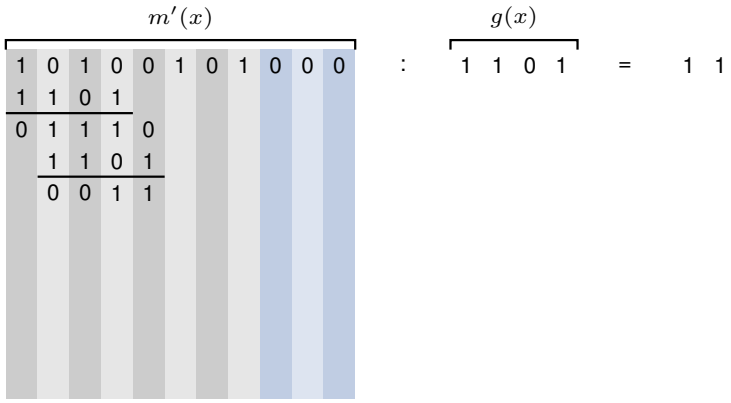
Beispiel: Generator $g(x) = x^3 + x^2 + 1$, Daten $m(x) = x^7 + x^5 + x^2 + 1$



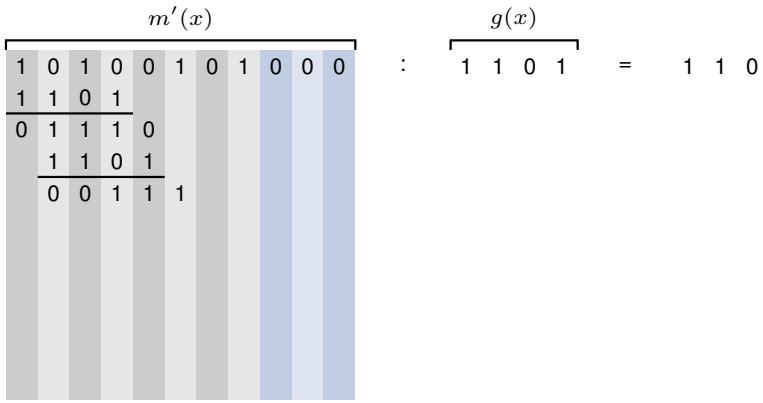
Beispiel: Generator $g(x) = x^3 + x^2 + 1$, Daten $m(x) = x^7 + x^5 + x^2 + 1$



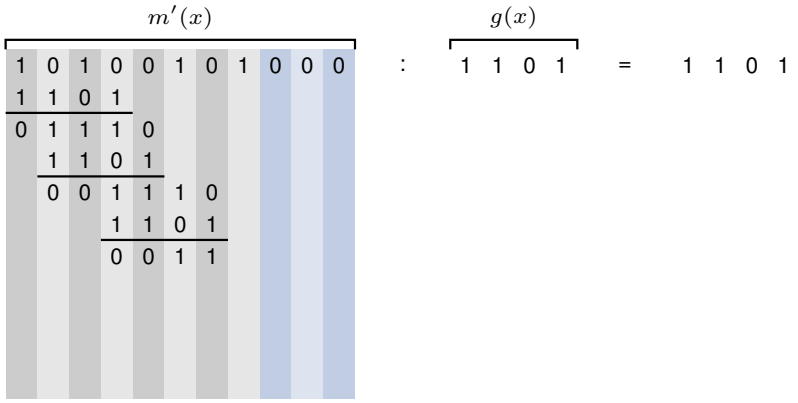
Beispiel: Generator $g(x) = x^3 + x^2 + 1$, Daten $m(x) = x^7 + x^5 + x^2 + 1$



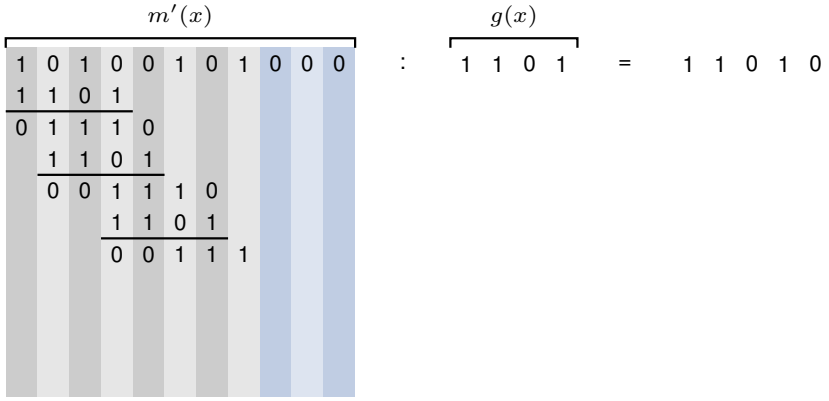
Beispiel: Generator $g(x) = x^3 + x^2 + 1$, Daten $m(x) = x^7 + x^5 + x^2 + 1$



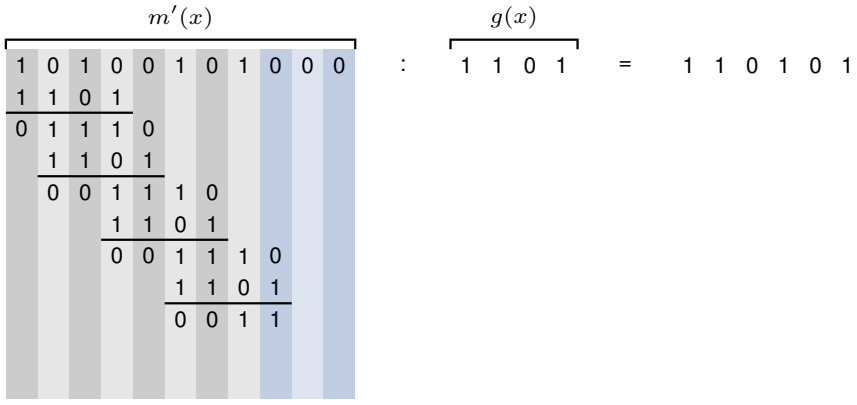
Beispiel: Generator $g(x) = x^3 + x^2 + 1$, Daten $m(x) = x^7 + x^5 + x^2 + 1$



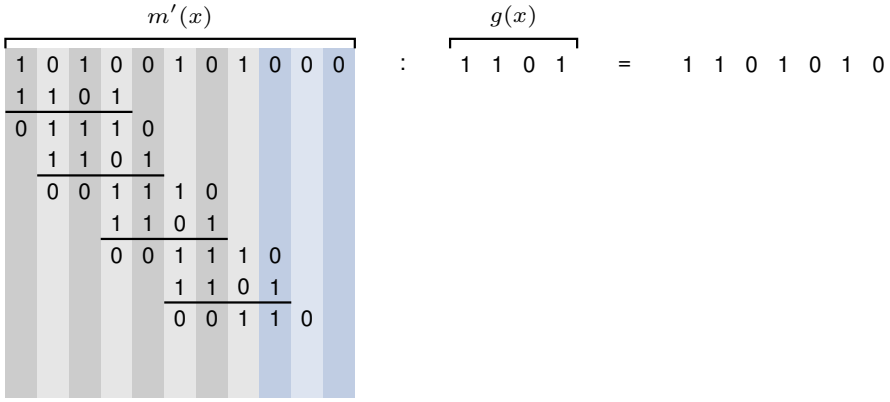
Beispiel: Generator $g(x) = x^3 + x^2 + 1$, Daten $m(x) = x^7 + x^5 + x^2 + 1$



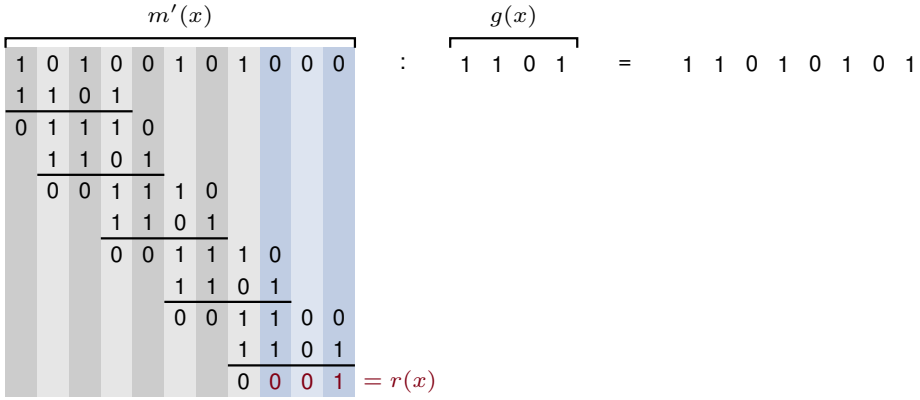
Beispiel: Generator $g(x) = x^3 + x^2 + 1$, Daten $m(x) = x^7 + x^5 + x^2 + 1$



Beispiel: Generator $g(x) = x^3 + x^2 + 1$, Daten $m(x) = x^7 + x^5 + x^2 + 1$



Beispiel: Generator $g(x) = x^3 + x^2 + 1$, Daten $m(x) = x^7 + x^5 + x^2 + 1$



Beispiel: Generator $g(x) = x^3 + x^2 + 1$, Daten $m(x) = x^7 + x^5 + x^2 + 1$

1. Koeffizienten bestimmen: $g(x) \mathrel{\hat{=}} 1101$ und $m(x) \mathrel{\hat{=}} 10100101$

2. $\text{grad}(g(x)) = 3 \Rightarrow$ Daten mit x^3 multiplizieren. Dies entspricht dem „Anhängen“ von 3 Nullen: $m'(x) = m(x) \cdot x^3 \mathrel{\hat{=}} 10100101000$

3. Polynomdivision $m'(x)/g(x)$ ausführen und den Rest $r(x)$ bestimmen.

Beispiel: Generator $g(x) = x^3 + x^2 + 1$, Daten $m(x) = x^7 + x^5 + x^2 + 1$

1. Koeffizienten bestimmen: $g(x) \mathrel{\hat{=}} 1101$ und $m(x) \mathrel{\hat{=}} 10100101$

2. $\text{grad}(g(x)) = 3 \Rightarrow$ Daten mit x^3 multiplizieren. Dies entspricht dem „Anhängen“ von 3 Nullen: $m'(x) = m(x) \cdot x^3 \mathrel{\hat{=}} 10100101000$

3. Polynomdivision $m'(x)/g(x)$ ausführen und den Rest $r(x)$ bestimmen.

4. Die zu sendende Nachricht ist $s(x) = m'(x) + r(x)$. Die Addition reduziert sich auf ein XOR, da wir auf GF(2) arbeiten.

$$\begin{array}{rcl} 1\ 0\ 1\ 0\ 0\ 1\ 0\ 1\ 0\ 0\ 0 & = & m'(x) \\ \oplus\quad\quad\quad\quad\quad\quad\quad\quad 0\ 0\ 1 & = & r(x) \\ \hline 1\ 0\ 1\ 0\ 0\ 1\ 0\ 1\ 0\ 0\ 1 & = & s(x) \end{array}$$

Beispiel: Generator $g(x) = x^3 + x^2 + 1$, Daten $m(x) = x^7 + x^5 + x^2 + 1$

1. Koeffizienten bestimmen: $g(x) \mathrel{\hat{=}} 1101$ und $m(x) \mathrel{\hat{=}} 10100101$
2. $\text{grad}(g(x)) = 3 \Rightarrow$ Daten mit x^3 multiplizieren. Dies entspricht dem „Anhängen“ von 3 Nullen: $m'(x) = m(x) \cdot x^3 \mathrel{\hat{=}} 10100101000$
3. Polynomdivision $m'(x)/g(x)$ ausführen und den Rest $r(x)$ bestimmen.
4. Die zu sendende Nachricht ist $s(x) = m'(x) + r(x)$. Die Addition reduziert sich auf ein XOR, da wir auf GF(2) arbeiten.

Der Empfänger prüft die Nachricht, indem er $r'(x) = (s(x) + e(x))/g(x)$ bestimmt, wobei $e(x)$ für mögliche Übertragungsfehler steht:

- $r'(x) \neq 0$ besagt, dass sicher ein Fehler aufgetreten ist
- $r'(x) = 0$ besagt, dass mit hoher Wahrscheinlichkeit kein Fehler aufgetreten ist

Welche Fehler erkennt CRC? Sei N die Länge der Checksumme, also $N = \text{grad}(g(x))$. Dann werden die folgenden Fehler erkannt:

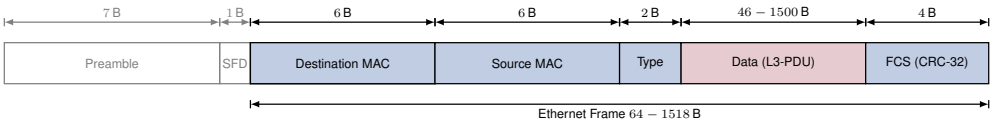
- Alle 1 bit-Fehler
- Isolierte 2 bit-Fehler, d. h. Fehler an den Bitstellen i und j wobei $i > j$
- Alle Burst-Fehler, deren Länge kleiner ist als N
- Einige Burst-Fehler, die länger sind als N

Welche Fehler erkennt CRC nicht zuverlässig oder gar nicht?

- Fehler, die länger sind als N
- Fehler, die aus mehreren Bursts bestehen
- Alle Fehler, die ein Vielfaches des Generatorpolynoms sind

2.4.2.2. Fallbeispiel: IEEE 802.3u (FastEthernet)

Frame vor der 4B5B-Kodierung:



- Präambel und Start Frame Delimiter (SFD) dienen der Taktsynchronisation
- Das erste Byte der Präambel wird durch das J/K-Symbol des 4B5B-Codes ersetzt (Start of Frame)
- Nach der Frame Check Sequence (FCS) wird das T/R-Symbol des 4B5B-Codes eingefügt (End of Frame)

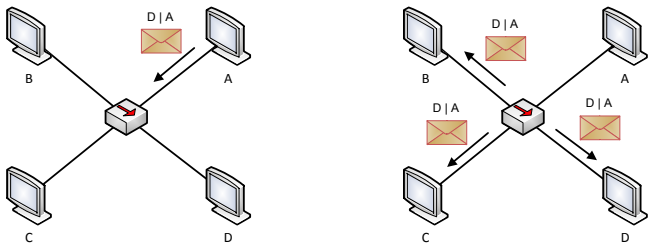
- Zwischen J/K und T/R liegende Daten werden gemäß des 4B5B-Codes kodiert
- Das Typfeld gibt die Art des Frames an
  (z. B. 0x0800 ≙ IPv4 Payload, 0x0806 ≙ ARP)
- Das Datenfeld muss (vor der Kodierung) mind. 46 B lang sein – andernfalls wird es bis zu diesem Wert gepadded

2.5. Verbindung auf Schicht 1 und 2

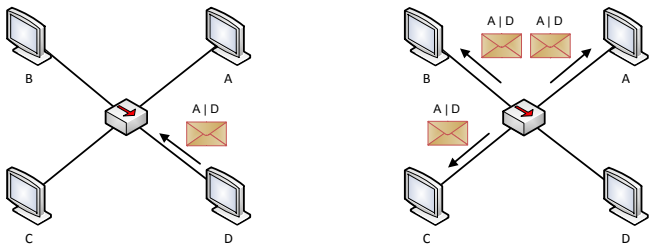
2.5.1. Hubs, Bridges und Switches

2.5.1.1. Verbindung auf Schicht 1: Hub [12]

- Knoten A sendet einen Rahmen an Knoten D
- Der Hub verbindet die einzelnen Links zu einem gemeinsamen Bus
- Der Rahmen erreicht alle Knoten
- Es darf folglich zu jedem Zeitpunkt nur ein Knoten senden, andernfalls treten Kollisionen auf



- Knoten D antwortet auf den Rahmen von A
- Auch die Antwort erreicht alle Knoten



Definition (Collision Domain):

Unter einer Kollisions-Domäne versteht man den Teil eines Direktverbindungsnetzes, innerhalb dem eine Kollision bei gleichzeitiger Übertragung mehrerer Knoten auftreten kann. Dieser wird häufig auch als Segment bezeichnet.

Sind Hubs mehr als nur Sternverteiler? Man unterscheidet aktive und passive Hubs:

- Aktive Hubs (Repeater) verstärken die Signale auf der physikalischen Schicht, ohne dabei die in Rahmen enthaltenen Felder wie Adressen oder Checksummen zu prüfen
- Passive Hubs sind wirklich nur Sternverteiler – man könnte genauso gut die einzelnen Adern der Patchkabel verlöten

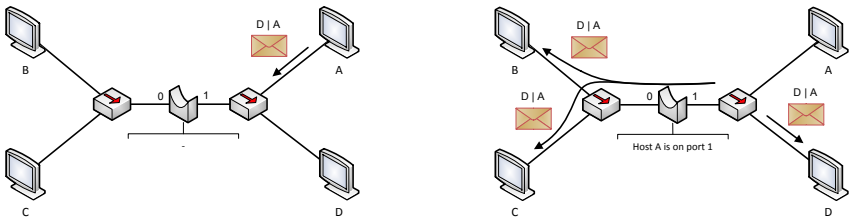
Kann man Hubs kaskadieren? Ja, aber es gilt bei Ethernet mit Baumtopologie (802.3a/i) die 5-4-3-Regel:

- Nicht mehr als 5 Abschnitte,
- verbunden durch 4 Repeater,
- wobei nur in 3 Abschnitten aktive Endgeräte enthalten sein dürfen.

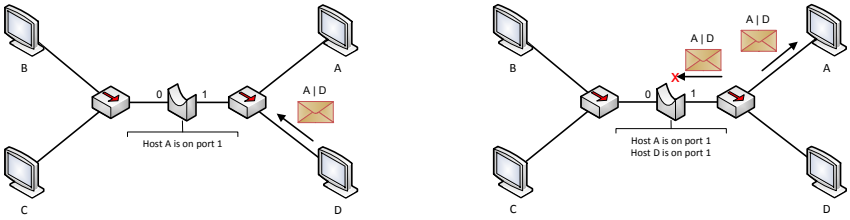
Jeder Abschnitt soll aufgrund der Dämpfung bei 802.3a (10BASE-2) nicht länger als 185 m sein, bei 802.3i (10BASE-T) nicht länger als 100 m zwischen Hub und Endgerät (Dämpfung). Aufgrund einer sicheren Kollissionserkennung ergibt sich bei 100BASE-TX eine maximale Ausdehnung von 500 m (→ Übung). **Können Hubs unterschiedliche Medientypen miteinander verbinden?**

- Ja, wenn auf allen Abschnitten dasselbe Medienzugriffsverfahren genutzt wird (beispielsweise Verbindung Ethernet über BNC- und Patch-Kabel mit jeweils gleicher Datenrate).
- Unterschiedliche Zugriffsverfahren können nicht gekoppelt werden.

2.5.1.2. Verbindung auf Schicht 2: Brücke [12]

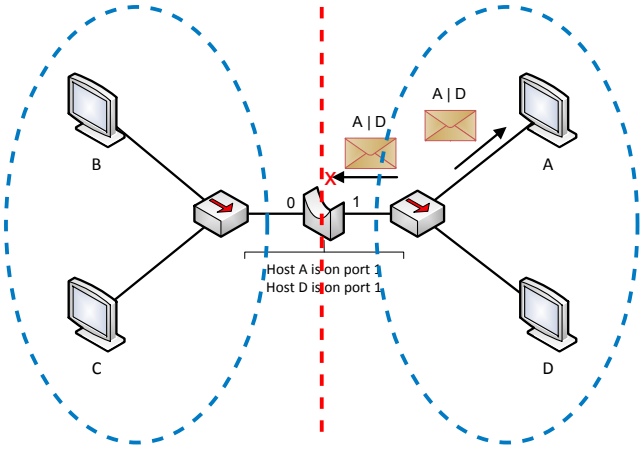


- Zwei Gruppen von Hosts, die jeweils über Hubs verbunden sind, werden im obigen Beispiel durch eine Brücke (engl. Bridge) gekoppelt
- Die Brücke arbeitet zunächst wie ein Hub mit 2 Ports (Learning-Phase)
- Dabei merkt sich die Brücke, über welchen Port ein Rahmen empfangen wurde
- So ordnet sie ihren beiden Ports 0 und 1 die MAC-Adressen der Knoten zu, die an den jeweiligen Port angeschlossen sind



2.5.1.3. Verbindung auf Schicht 2: Brücke

- Die Ziel-Adresse eingehender Rahmen wird mit den Einträgen in der Switching-Table verglichen
- Ist ein Eintrag vorhanden, wir der Rahmen nur an den betreffenden Ziel-Port weitergeleitet
- Ist kein Eintrag vorhanden, so wird der Rahmen an alle Ports weitergeleitet
- Einträge erhalten einen Zeitstempel (Timestamp) und werden nach einem festen Zeitintervall invalidiert
- Eine Brücke unterbricht Kollisionsdomänen (auch als Segmente bezeichnet)
- Wenn die Brücke alle angeschlossenen Geräte kennt, darf in jedem der beiden Segmente jeweils ein Knoten zur selben Zeit senden

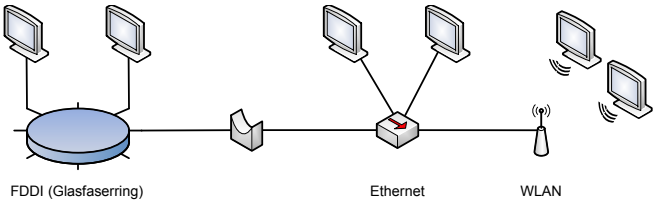


Brücken können auch genutzt werden, um Netzsegmente mit unterschiedlichen Zugriffsverfahren zu koppeln. Derartige Brücken bezeichnet man auch als Translation Bridges:

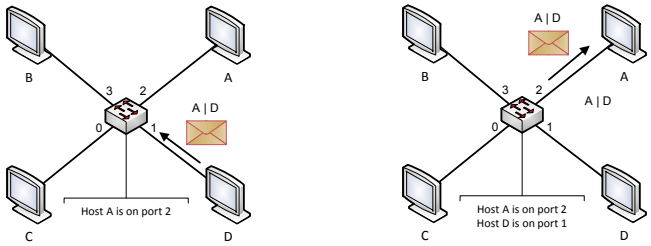
- FDDI-Ethernet Bridge zwischen Token Passing und CSMA/CD
- WLAN Access Point zwischen CSMA/CD und CSMA/CA

Diese Kopplung ist transparent, d. h.

- angeschlossene Stationen bemerken nicht, dass eine Brücke verwendet wird und
- im normalen Betrieb wird ein Host niemals direkt mit der Brücke kommunizieren. (Ausnahme: Management-Anwendung kommuniziert mit Brücke)



Voraussetzung für Translation Bridges: Die MAC-Adressen müssen „kompatibel“ sein, um Empfänger über ihre MAC-Adressen identifizieren zu können.
Switches – Brücken mit mehreren Ports Ähnlich wie Brücken speichern Switches in einer Tabelle, welche Hosts an welchen Port angeschlossen sind: Mit Switches ist ein



kollisionsfreier Betrieb auch bei Ethernet möglich:

- An jedem Port darf höchstens ein Host (oder ein anderes Switch) angeschlossen sein
- Die Hosts müssen Vollduplex unterstützen

Anmerkungen

- Brücken und Switches sind für Hosts transparent, d. h. ein Host weiß nicht, dass er über eine Brücke mit anderen Hosts kommuniziert.
- Sender- und Empfänger-Adresse werden von Brücken nicht verändert
- Brücken schränken nicht die Erreichbarkeit innerhalb des Direktverbindungsnetzes ein
- Ein Broadcast (MAC-Adresse ff:ff:ff:ff:ff:ff) wird von allen Hosts empfangen (man spricht daher auch von Broadcast-Domänen im Unterschied zu einer Kollisions-Domäne)
- Brücken und Switches benötigen zur Erfüllung ihrer grundlegenden Aufgaben keine eigene Adresse
- Weiterleitungsentscheidungen werden aber auf Basis der Ziel-Adresse und der aktuellen Switching-Tabelle getroffen

Ferner unterscheidet man zwischen zwei unterschiedlichen Switching-Arten:

- Store-and-Forward: Eingehende Rahmen werden vollständig empfangen und deren FCS geprüft. Falls der Ausgangsport belegt ist, kann eine begrenzte Anzahl von Rahmen gepuffert werden.
- Cut-Through: Beginne mit der Serialisierung des Rahmens, sobald der Ausgangsport bestimmt wurde. Die FCS wird in diesem Fall nicht geprüft.

2.5.1.4. Schleifen auf Schicht 2

Problembeschreibung

- Schleifen auf Schicht 1 bedeuten Kurzschluss.
- Schleifen auf Schicht 2 führen dazu, dass mehrere Kopien eines Rahmens erzeugt werden und im Netzwerk zirkulieren.

Wie entstehen Schleifen?

- Auch wenn Direktverbindungsnetze räumlich begrenzt sind, kann man schnell den Überblick verlieren und ungewollt Schleifen erzeugen.
- Manchmal erzeugt man Schleifen auch absichtlich, so dass redundante Pfade entstehen. Fällt eine Verbindung aus, kann der Verkehr umgeleitet werden.

Wie werden Schleifen vermieden?

- Switches unterstützen das sog. Spanning Tree Protocol (STP)
- Ziel ist die Deaktivierung redundanter Pfade, so dass alle Netzsegmente schleifenfrei erreichbar sind
- Fällt eine Verbindung aus, wird ggf. einer dieser Pfade reaktiviert

2.5.1.5. Zusammenfassung

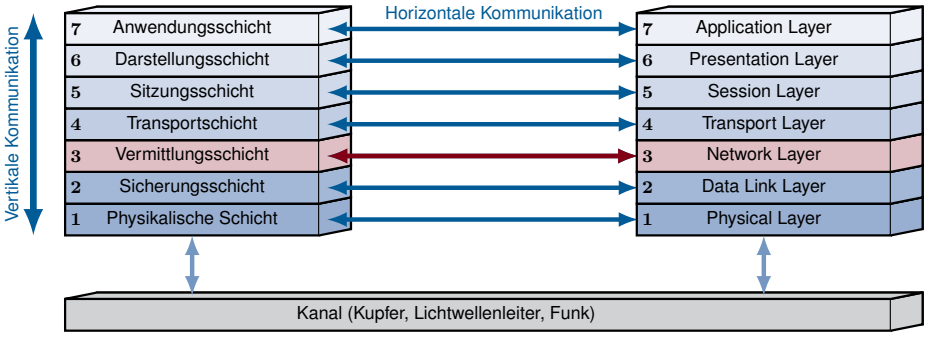
Wir sollten wissen,

- wie Netzwerke als Graphen dargestellt werden können,
- was der Unterschied zwischen einem MST und einem SPT ist,
- welche unterschiedlichen Medienzugriffsverfahren es gibt,
- wie diese Kollisionen vermeiden oder mit ihnen umgehen,
- warum die maximale Länge eines Ethernet-Segments 500 m beträgt,
- wie Knoten in Direktverbindungsnetzen adressiert werden,
- wie MAC-Adressen bei Ethernet aufgebaut sind,
- wie mehrere Direktverbindungsnetze zu einem größeren miteinander verbunden werden können,
- worin der Unterschied zwischen Hubs, Bridges und Switches besteht,
- wie Switches lernen, an welchem Port, welche Geräte angeschlossen sind und wie Weiterleitungsentscheidungen getroffen werden und
- was eine Kollisions-Domäne bzw. eine Broadcast-Domäne ist.

3. Vermittlungsschicht

3.1. Motivation

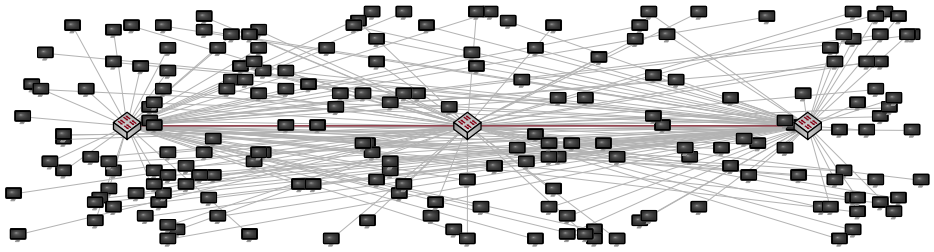
3.1.0.6. Einordnung im ISO/OSI-Modell



3.1.0.7. Motivation

Sind Direktverbindungsnetze wie Ethernet skalierbar?

- Alle angeschlossenen Hosts sind direkt bzw. über wenige Switches erreichbar
- MAC-Adressen bieten keine logische Struktur zur Adressierung
- Gruppierung von Geräten in kleinere Netze (Subnetze) durch MAC-Adressen nicht unterstützt



Aufgaben der Vermittlungsschicht:

- Kopplung unterschiedlicher Direktverbindungsnetze
- Strukturierte Aufteilung in kleinere Subnetze
- Logische und global eindeutige Adressierung von Geräten
- Wegwahl zwischen Geräten über mehrere Hops hinweg

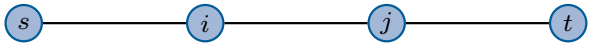
3.2. Vermittlungsarten

3.2.0.8. Vermittlungsarten

Es gibt drei grundlegende Vermittlungsarten:

- **Leitungsvermittlung**
  „Reserviere eine dedizierte Leitung zwischen Sender und Empfänger“
- **Nachrichtenvermittlung**
  „Wähle für jede Nachricht individuell einen Weg“
- **Paketvermittlung**
  „Teile eine Nachricht in mehrere kleinere Pakete auf und versende jedes Paket unabhängig von den anderen“

Im Folgenden charakterisieren wir diese drei Vermittlungsarten anhand des Beispielnetzwerks



mit $n = 2$ Vermittlungsknoten hinsichtlich der Gesamtdauer T einer Übertragung von l Datenbits über die Distanz d und motivieren so die Vorteile der Paketvermittlung.

3.2.1. Leitungsvermittlung

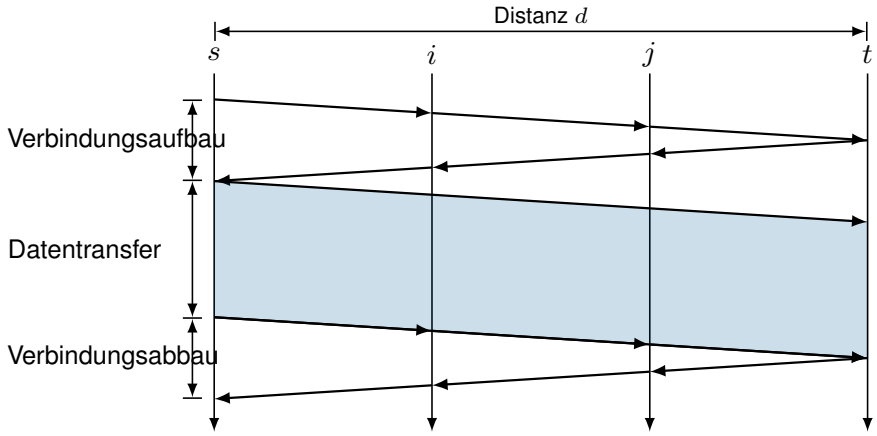
Leitungsvermittlung Während einer verbindungsorientierten Übertragung können drei Phasen unterschieden werden:

1. **Verbindungsaufbau**
   - Austausch von Signalisierungsnachrichten zum Aufbau einer dedizierten Verbindung zwischen Sender und Empfänger.
   - Dieser Schritt beinhaltet die Wegwahl, welche vor Beginn der Datenübertragung durchgeführt wird.
2. **Datenaustausch**
   - Kanal steht den Kommunikationspartnern zur exklusiven Nutzung bereit.
   - Auf die Adressierung des Kommunikationspartners kann während der Übertragung weitgehend verzichtet werden (Punkt-zu-Punkt-Verbindung).
3. **Verbindungsabbau**
   - Austausch von Signalisierungsnachrichten zum Abbau der Verbindung.
   - Die durch die Verbindung belegten Ressourcen werden für nachfolgende Verbindungen freigegeben.

Übertragungszeit bei Leitungsvermittlung Wir nehmen an, dass

- die Serialisierungszeit von Signalisierungsnachrichten vernachlässigbar klein ist,
- die Verarbeitungszeiten und Wartezeiten in jedem Knoten vernachlässigbar klein sind und dass

- der Sender s einen Datenblock der Länge L an einem Stück übertragen möchte.



$$T_{LV} = 2t_p + t_s + 2t_p = t_s + 4t_p = \frac{L}{r} + 4 \cdot \frac{d}{\nu c}$$

Vorteile der Leitungsvermittlung

- Gleichbleibende Güte der dedizierten Verbindung nach dem Verbindungsaufbau
- Schnelle Datenübertragung ohne Notwendigkeit, weitere Vermittlungsentscheidungen treffen zu müssen

Nachteile der Leitungsvermittlung

- Ressourcenverschwendung, da Leitung zur exklusiven Nutzung reserviert wird
- Verbindungsaufbau kann komplex sein und benötigt u. U. weit mehr Zeit, als die Ausbreitungsverzögerungen vermuten lassen (z. B. Einwahl ins Internet mittels Modem)
- Hoher Aufwand bei der Schaltung physikalischer Verbindungen

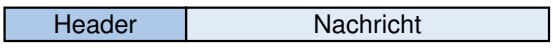
Einsatz in heutigen Netzwerken

- Leitungsvermittlung wird häufig durch Paketvermittlung ersetzt (z. B. Voice over IP)
- In vielen Vermittlungsnetzen wird Leitungsvermittlung zumindest virtualisiert in Form von Virtual Circuits unterstützt (z. B. Frame Relay, ATM, MPLS)

3.2.2. Nachrichtenvermittlung

Nachrichtenvermittlung Veränderungen bei der Nachrichtenvermittlung:

- Aufbau und Abbau einer dedizierten Verbindung entfallen
- Der gesamten Nachricht der Länge L wird ein Header der Länge L_H vorangestellt



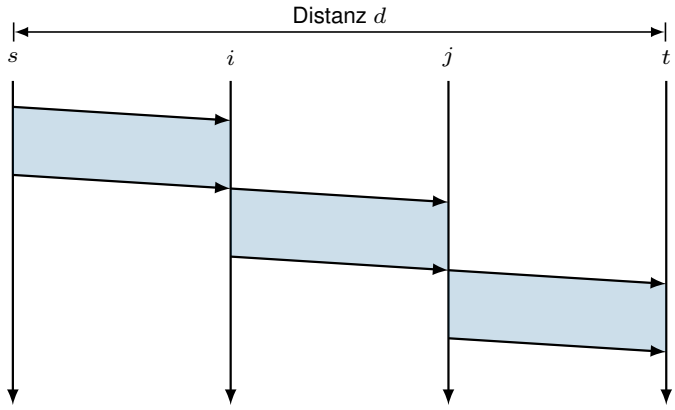
- Der Header beinhaltet insbesondere Adressinformationen, die geeignet sind, Sender und Empfänger auch über mehrere Zwischenstationen hinweg eindeutig zu identifizieren

- Die so entstehende PDU wird als Ganzes übertragen

Hoffnung: Zeitersparnis, da die Phasen zum Aufbau und Abbau der Verbindung entfallen.
Analogie: Post / DHL / Paketdienste

- Absender verpackt Ware und versieht das Paket mit Adressinformationen (Header)
- Die Adressen identifizieren Absender und Empfänger weltweit eindeutig und haben eine logische Struktur, die eine effiziente Zuordnung im Transportnetz der Post erlaubt

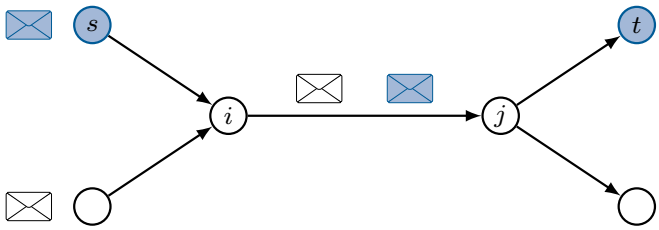
Übertragungszeit bei Nachrichtenvermittlung Erinnerung: Hier $n = 2$ Vermittlungsknoten i und j.



$$T_{NV} = (n+1) \cdot t_s + t_p = (n+1) \cdot \frac{L_H + L}{r} + \frac{d}{\nu c}$$

Multiplexing auf Nachrichtenebene

- Das Wegfallen fest vorgegebener Pfade ermöglicht die gemeinsame Nutzung von Teilstrecken
- Dies entspricht dynamischem Zeitmultiplex (Time Division Multiplex, TDM)



Vorteile:

- Flexibles Zeitmultiplex von Nachrichten
- Bessere Ausnutzung der Kanalkapazität
- Keine Verzögerung beim Senden des ersten Pakets durch Verbindungsaufbau

Nachteile:

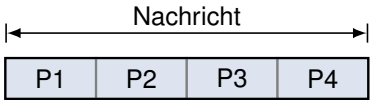
- Pufferung von Nachrichten, wenn (i, j) ausgelastet
- Verlust von Nachrichten durch begrenzten Puffer möglich (Stausituation → Kapitel 4)
- Mehrfache Serialisierung der ganzen Nachricht

3.2.3. Paketvermittlung

3.2.3.1. Paketvermittlung

Unterschiede zur Nachrichtenvermittlung:

- Nachrichten werden nicht mehr als Einheit übertragen sondern in kleinere Pakete unterteilt:

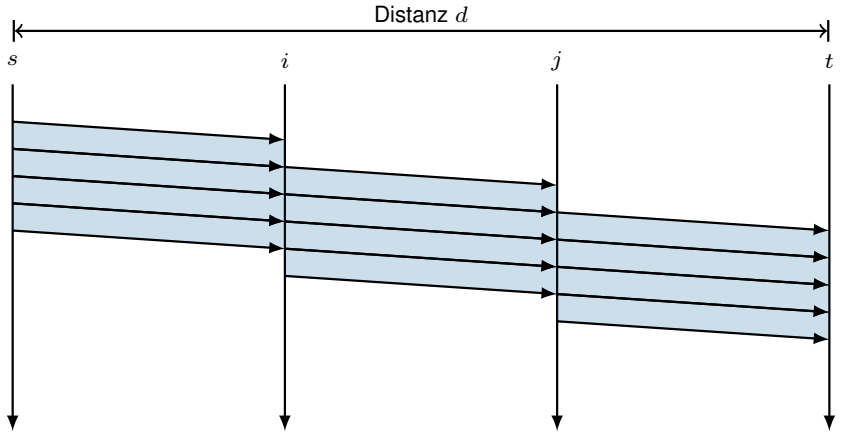


- Jedem Paket wird ein eigener Header vorangestellt, der alle Informationen zur Weiterleitung und ggf. auch zur Reassemblierung enthält:

- Pakete werden unabhängig voneinander vermittelt, d. h. Pakete derselben Nachricht können über unterschiedliche Wege zum Empfänger gelangen.
- Im Allgemeinen müssen die einzelnen Pakete nicht gleich groß sein, es gibt aber Anforderungen an die minimale ($p_{\min}$) und maximale ($p_{\max}$) Paketgröße.

Übertragungszeit bei Paketvermittlung

- Erinnerung: Hier $n = 2$ Vermittlungsknoten i und j.
- Vereinfachend nehmen wir an, dass die Nachrichtenlänge ein Vielfaches von $p_{\max}$ ist. ($\Rightarrow$ alle Pakete haben dieselbe Größe $p_{\max}$).

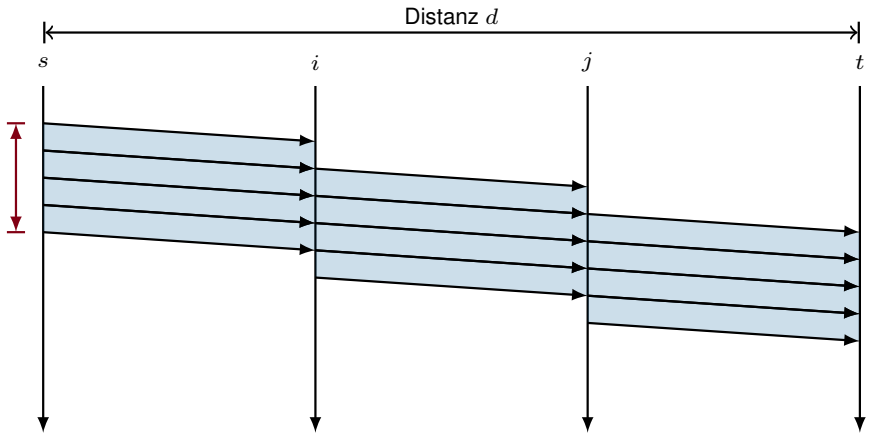


$$T_{PV} = \frac{1}{r}\left(\left\lceil \frac{L}{p_{\max}} \right\rceil \cdot L_h + L\right) + \frac{d}{\nu c} + n \cdot \frac{L_h + p_{\max}}{r}$$

Übertragungszeit bei Paketvermittlung

- Erinnerung: Hier $n = 2$ Vermittlungsknoten i und j.

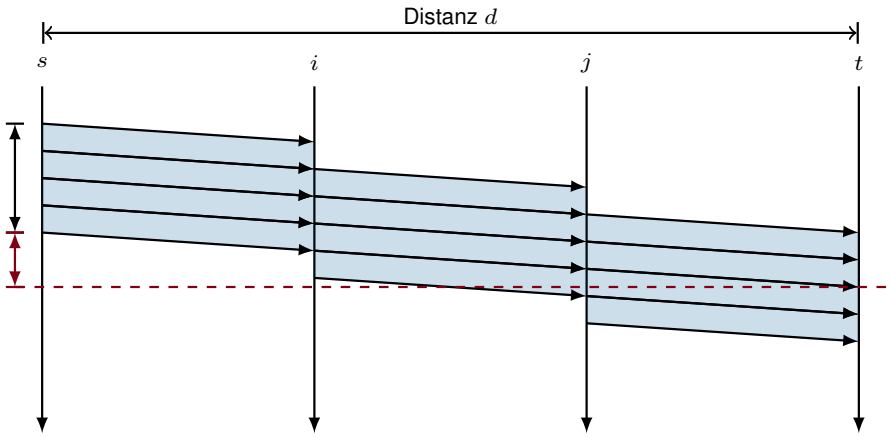
- Vereinfachend nehmen wir an, dass die Nachrichtenlänge ein Vielfaches von $p_{\max}$ ist. ($\Rightarrow$ alle Pakete haben dieselbe Größe $p_{\max}$).



$$T_{PV} = \frac{1}{r}\left(\left\lceil \frac{L}{p_{\max}} \right\rceil \cdot L_h + L\right) + \frac{d}{\nu c} + n \cdot \frac{L_h + p_{\max}}{r}$$

Übertragungszeit bei Paketvermittlung

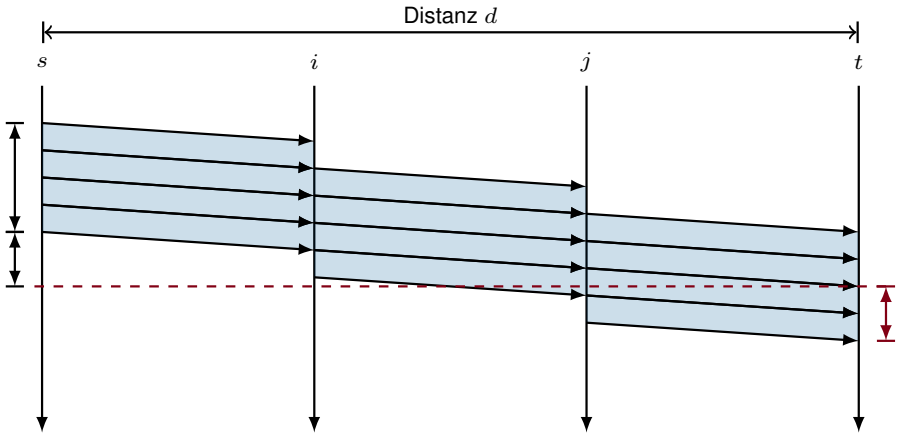
- Erinnerung: Hier $n = 2$ Vermittlungsknoten i und j.
- Vereinfachend nehmen wir an, dass die Nachrichtenlänge ein Vielfaches von $p_{\max}$ ist. ($\Rightarrow$ alle Pakete haben dieselbe Größe $p_{\max}$).



$$T_{PV} = \frac{1}{r}\left(\left\lceil \frac{L}{p_{\max}} \right\rceil \cdot L_h + L\right) + \frac{d}{\nu c} + n \cdot \frac{L_h + p_{\max}}{r}$$

Übertragungszeit bei Paketvermittlung

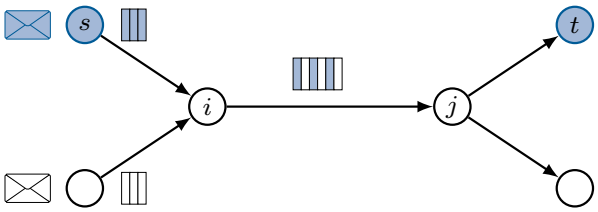
- Erinnerung: Hier $n = 2$ Vermittlungsknoten i und j.
- Vereinfachend nehmen wir an, dass die Nachrichtenlänge ein Vielfaches von $p_{\max}$ ist. ($\Rightarrow$ alle Pakete haben dieselbe Größe $p_{\max}$).



$$T_{PV} = \frac{1}{r}\left(\left\lceil \frac{L}{p_{\max}} \right\rceil \cdot L_h + L\right) + \frac{d}{\nu c} + n \cdot \frac{L_h + p_{\max}}{r}$$

Multiplexing auf Paketebene

- Durch die Vermittlung kleiner Pakete statt langer Nachrichten werden Engpässe fairer genutzt
- Gehen Pakete verloren, müssen nur Teile einer größeren Nachricht wiederholt werden



Vorteile:

- Flexibles Zeitmultiplex einzelner Pakete
- Pufferung kleiner Pakete statt ganzer Nachrichten

Nachteile:

- Verlust von Paketen durch begrenzten Puffer möglich
- Jedes Paket benötigt seinen eigenen Header (Overhead)
- Empfänger muss Pakete wieder zusammensetzen

Vergleich der drei Verfahren

$$\begin{aligned}
\text{Leitungsvermittlung:} \quad & T_{LV} = \frac{L}{r} + 4\frac{d}{\nu c} \\
\text{Nachrichtenvermittlung:} \quad & T_{NV} = (n+1)\frac{L_h + L}{r} + \frac{d}{\nu c} \\
\text{Paketvermittlung:} \quad & T_{PV} = \frac{1}{r}\left(\left\lceil \frac{L}{p_{\max}} \right\rceil \cdot L_h + L\right) + \frac{d}{\nu c} + n \cdot \frac{L_h + p_{\max}}{r}
\end{aligned}$$

Zahlenbeispiel:

- Die Distanz zwischen Sender und Empfänger beträgt $d = 1000\,\text{km}$
- Für Glasfaserleitungen beträgt $\nu = 0.9$
- Es kommen $n = 5$ Vermittlungsknoten zum Einsatz
- Die Datenrate beträgt auf allen Teilstrecken $r = 1\,\text{Mbit/s}$
- Die Länge der zu sendenden Nachricht beträgt $L = 5\,\text{MiB}$
- Die maximale Paketgröße betrage $p_{\max} = 1480\,\text{B}$
- Die Headergröße pro Nachricht / Paket betrage $L_h = 20\,\text{B}$

Es ergeben sich folgende Zahlenwerte:

$$T_{LV} \approx 42\,\text{s},\ T_{NV} \approx 5033\,\text{s} \text{ und } \ T_{PV} \approx 43\,\text{s}.$$

Wo werden die Verfahren eingesetzt? Leitungsvermittlung:

- Analoge Telefonverbindungen (POTS)
- Interneteinwahl („letzte Meile“)
- Standortvernetzung von Firmen
- Virtuelle Kanäle (engl. Virtual Circuits) in unterschiedlichen Arten von Vermittlungsnetzen (Frame Relay, ATM, MPLS, . . .)

Nachrichtenvermittlung:

- Kaum praktische Anwendung auf Schicht 3
- Aber: Nachrichtenvermittlung existiert aus Sicht höherer Schichten (ab Schicht 5 aufwärts)

Paketvermittlung:

- In den meisten modernen Datennetzen
- Zunehmend auch zur Sprachübertragung (Voice over IP)
- Digitales Radio / Fernsehen
- Viele Peripherieschnittstellen an Computern (PCI, USB, Thunderbolt)

3.3. Adressierung im Internet

3.3.0.2. Adressierung im Internet

Die Sicherungsschicht (Schicht 2) bietet

- mehr oder weniger fairen Medienzugriff bei von mehreren Hosts geteilten Medien,
- einen „ausreichenden“ Schutz vor Übertragungsfehler und

- Adressierung innerhalb eines Direktverbindungsnetzes.

Die Vermittlungsschicht (Schicht 3) ergänzt dies um

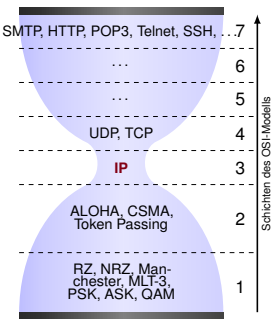
- Möglichkeiten zur global eindeutigen und strukturierten / logischen Adressierung sowie
- Verfahren zur Bestimmung von (möglichst) optimalen Pfaden.

Wir beschränken uns in diesem Teilkapitel auf die Betrachtung von

- IP (Internet Protocol, 1981) bzw.
- seinem Nachfolger IPv6 (1998).

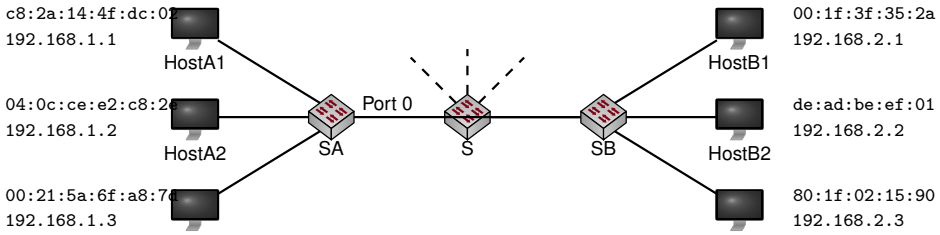
Beispiele für alternative Protokolle der Netzwerkschicht:

- IPX (Internetwork Packet Exchange, 1990)
- DECnet Phase 5 (1987)
- AppleTalk (1983)

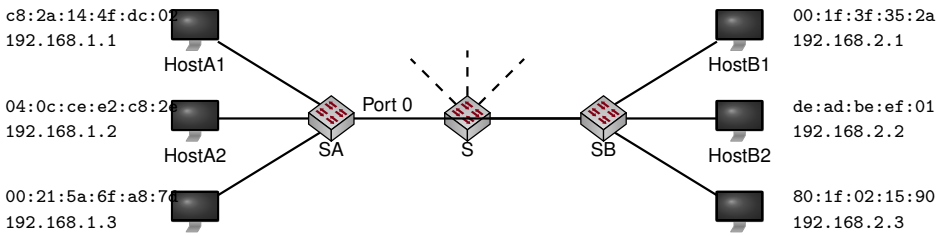


3.3.1. Internet Protocol (IP)

Internet Protocol (IPv4) [6] Wir betrachten das folgende Beispielnetz, welches auf einem aktuellen Ethernet-Standard basiert:



Wie viele Einträge enthält die Switching-Tabelle von Switch SA? **Internet Protocol (IPv4) [6]** Wir betrachten das folgende Beispielnetz, welches auf einem aktuellen Ethernet-Standard basiert:



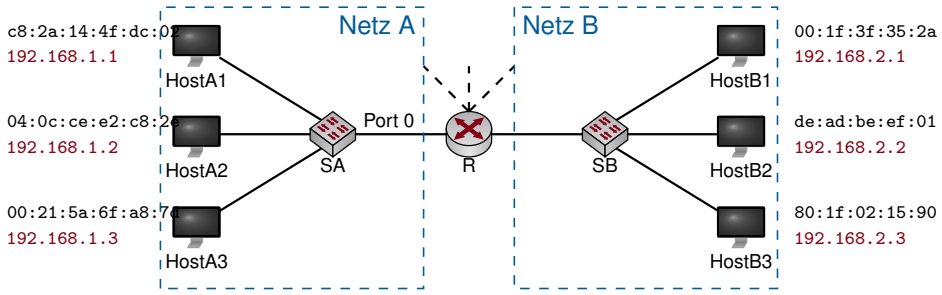
Wie viele Einträge enthält die Switching-Tabelle von Switch SA?

- Genau einen Entrag für jeden bekannten Host
- Die meisten MAC-Adressen werden auf Port 0 abgebildet
- Eine Zusammenfassung von Einträgen ist i. A. nicht möglich, da MAC-Adressen nicht die Position eines Knotens innerhalb eines Netzwerks wiederspiegeln

Port	MAC
0	00:1f:3f:35:2a:84
0	de:ad:be:ef:01:01
0	80:1f:02:15:90:ea
1	c8:2a:14:4f:dc:02
2	04:0c:ce:e2:c8:2e
3	00:21:5a:6f:a8:7d
⋮	⋮

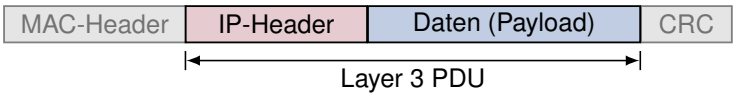
⇒ Es erscheint sinnvoll, von physikalischen Adressen zu abstrahieren

Wir betrachten das Beispielnetz mit einem Router (R) in der Mitte:



- Jedem Host ist eine IP-Adresse zugewiesen. Jede IP-Adresse ist in vier Gruppen zu je einem Byte, durch Punkte getrennt, dargestellt (Dotted Decimal Notation).
- In diesem Beispiel identifiziert das 4. Oktett einen Host innerhalb eines Netzes.
- Die ersten drei Oktette identifizieren das Netzwerk, in dem sich der Host befindet.
- Der Router R trifft Weiterleitungsentscheidungen auf Basis der Ziel-IP-Adresse.

⇒ Jedes Paket muss mit einer Absender- und Ziel-IP-Adresse (im IP-Header) versehen werden:



IP-Header

- Der IP-Header beinhaltet nicht nur Quell- und Ziel-Adresse.
- Die Verwendung der wichtigsten Felder wird später in diesem Kapitel anhand von Beispielen genauer erläutert werden.

Version

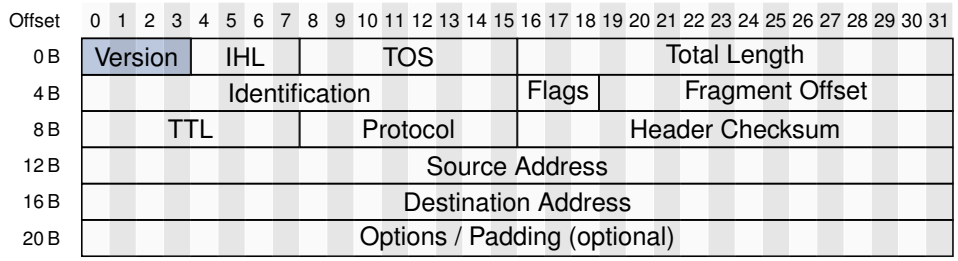


Abbildung 3.1.: IPv4-Header (minimale Länge: 20 Byte)

- Gibt die verwendete IP-Version an.
- Gültige Werte sind 4 (IPv4) und 6 (IPv6).

IP-Header

- Der IP-Header beinhaltet nicht nur Quell- und Ziel-Adresse.
- Die Verwendung der wichtigsten Felder wird später in diesem Kapitel anhand von Beispielen genauer erläutert werden.

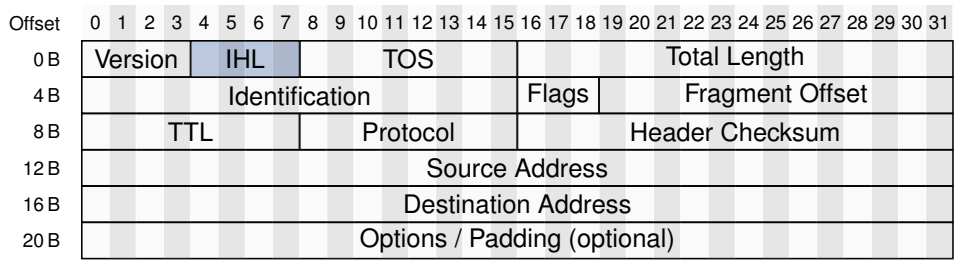


Abbildung 3.2.: IPv4-Header (minimale Länge: 20 Byte)

IHL (IP Header Length)

- Gibt die Länge des IP Headers inkl. Optionen in Vielfachen von 32 Bit an.
- Wichtig, da der IPv4-Header durch Optionsfelder variable Länge hat.

IP-Header

- Der IP-Header beinhaltet nicht nur Quell- und Ziel-Adresse.
- Die Verwendung der wichtigsten Felder wird später in diesem Kapitel anhand von Beispielen genauer erläutert werden.

TOS (Type of Service)

- Dient der Klassifizierung und Priorisierung von IP-Paketen (z. B. Hinweis auf zeitsensitive Daten wie Sprachübertragungen).
- Möglichkeit zur Staukontrolle (Explicit Congestion Notification) auf Schicht 3 (optional).

IP-Header

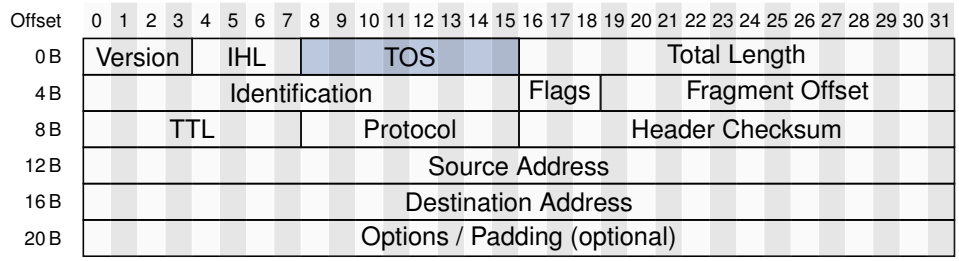


Abbildung 3.3.: IPv4-Header (minimale Länge: 20 Byte)

- Der IP-Header beinhaltet nicht nur Quell- und Ziel-Adresse.
- Die Verwendung der wichtigsten Felder wird später in diesem Kapitel anhand von Beispielen genauer erläutert werden.

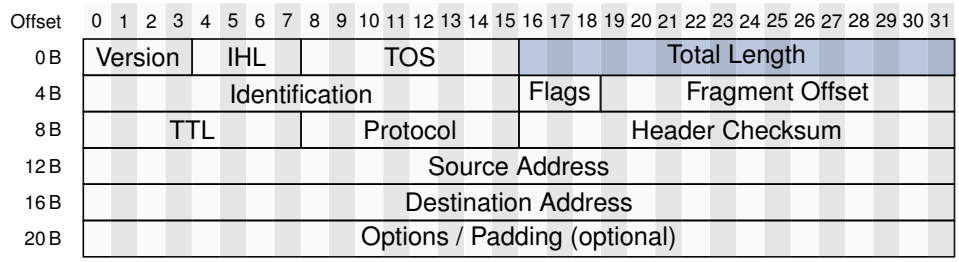


Abbildung 3.4.: IPv4-Header (minimale Länge: 20 Byte)

Total Length

- Gibt die Gesamtlänge des IP-Pakets (Header + Daten) in Bytes an.
- Die Maximallänge eines IP-Pakets beträgt damit 65535 Byte.
- Der Sender passt die Größe ggf. an, um Fragmentierung zu vermeiden.
- Die maximale Paketlänge, so dass keine Fragmentierung notwendig ist, bezeichnet man als Maximum Transmission Unit (MTU). Diese ist Abhängig von Schicht 2/1 und beträgt bei FastEthernet 1500 Byte.

IP-Header

- Der IP-Header beinhaltet nicht nur Quell- und Ziel-Adresse.
- Die Verwendung der wichtigsten Felder wird später in diesem Kapitel anhand von Beispielen genauer erläutert werden.

Identification

- Für jedes IP-Paket (zufällig) gewählter 16 Bit langer Wert.
- Dient der Identifikation zusammengehörender Fragmente (IP-Fragmentierung → später).

IP-Header

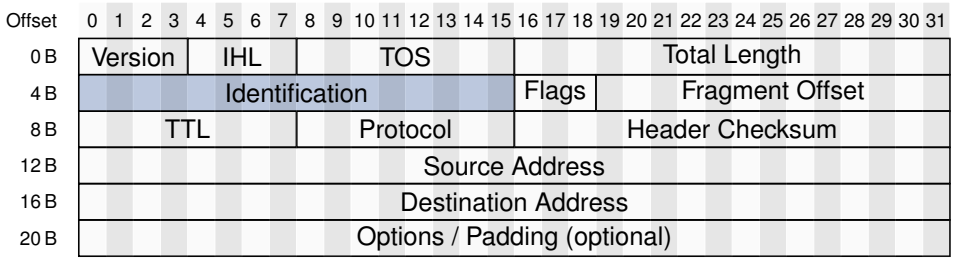


Abbildung 3.5.: IPv4-Header (minimale Länge: 20 Byte)

- Der IP-Header beinhaltet nicht nur Quell- und Ziel-Adresse.
- Die Verwendung der wichtigsten Felder wird später in diesem Kapitel anhand von Beispielen genauer erläutert werden.

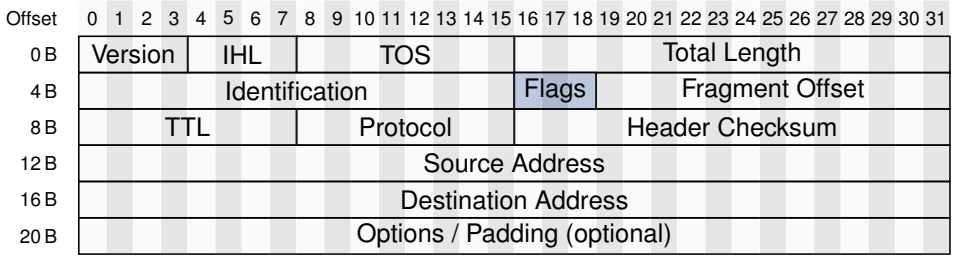


Abbildung 3.6.: IPv4-Header (minimale Länge: 20 Byte)

Flags

- Bit 16: Reserviert und wird auf 0 gesetzt.
- Bit 17: Don't Fragment (DF). Ist dieses Bit 1, so darf das IP-Paket nicht fragmentiert werden.
- Bit 18: More Fragments (MF). Gibt an, ob weitere Fragmente folgen (1) oder dieses Paket das letzte Fragment ist (0). Wurde das Paket nicht fragmentiert, wird es ebenfalls auf 0 gesetzt.

IP-Header

- Der IP-Header beinhaltet nicht nur Quell- und Ziel-Adresse.
- Die Verwendung der wichtigsten Felder wird später in diesem Kapitel anhand von Beispielen genauer erläutert werden.

Fragment Offset

- Gibt die absolute Position der Daten in diesem Fragment bezogen auf das unfragmentierte Paket in ganzzahligen Vielfachen von 8 Byte an.
- Ermöglicht zusammen mit dem Identifier und MF-Bit die Reassemblierung fragmentierter Pakete in der richtigen Reihenfolge.

IP-Header

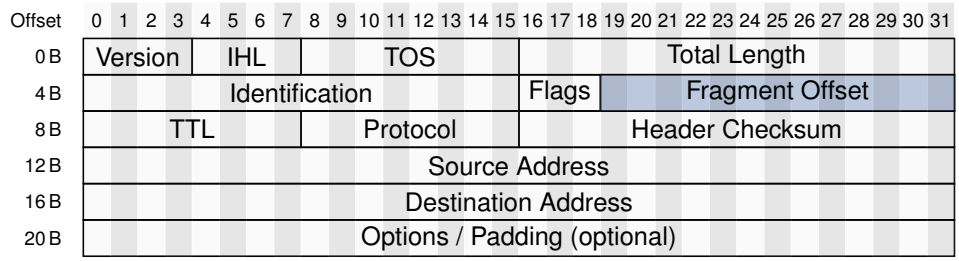


Abbildung 3.7.: IPv4-Header (minimale Länge: 20 Byte)

- Der IP-Header beinhaltet nicht nur Quell- und Ziel-Adresse.
- Die Verwendung der wichtigsten Felder wird später in diesem Kapitel anhand von Beispielen genauer erläutert werden.

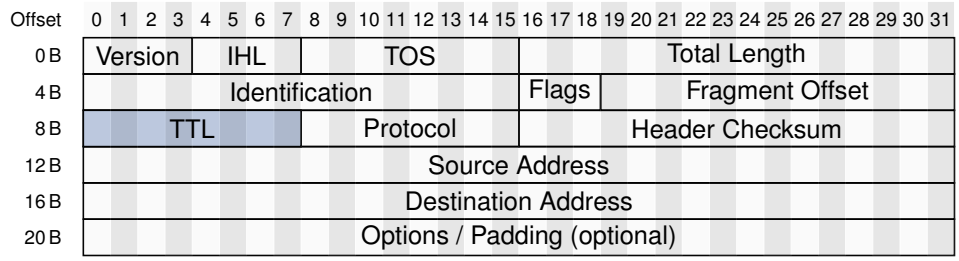


Abbildung 3.8.: IPv4-Header (minimale Länge: 20 Byte)

TTL (Time to Live)

- Leitet ein Router ein IP-Paket weiter, so dekrementiert er das TTL-Feld um 1.
- Erreicht das TTL-Feld den Wert 0, so verwirft ein Router das Paket und sendet eine Benachrichtigung an den Absender (ICMP Time Exceeded → später).
- Dieser Mechanismus beschränkt die Pfadlänge im Internet und verhindert endlos kreisende Pakete infolge von Routing Loops.

IP-Header

- Der IP-Header beinhaltet nicht nur Quell- und Ziel-Adresse.
- Die Verwendung der wichtigsten Felder wird später in diesem Kapitel anhand von Beispielen genauer erläutert werden.

Protocol

- Identifiziert das Protokoll auf Schicht 4, welches in der Payload (Daten) des IP-Pakets enthalten ist.
- Relevant u. a. für das Betriebssystem, um Pakete dem richtigen Prozess zuordnen zu können.
- Gültige Werte sind beispielsweise `0x06` (TCP) und `0x08` (UDP).

IP-Header

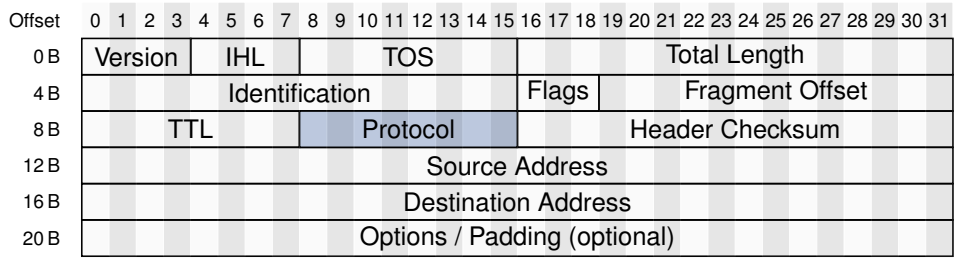


Abbildung 3.9.: IPv4-Header (minimale Länge: 20 Byte)

- Der IP-Header beinhaltet nicht nur Quell- und Ziel-Adresse.
- Die Verwendung der wichtigsten Felder wird später in diesem Kapitel anhand von Beispielen genauer erläutert werden.

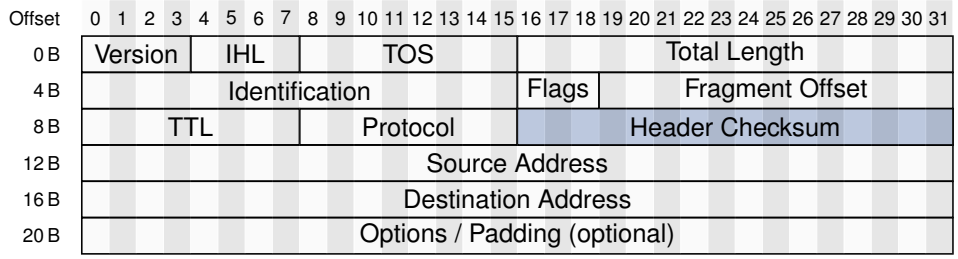


Abbildung 3.10.: IPv4-Header (minimale Länge: 20 Byte)

Header Checksum

- Einfache, auf Geschwindigkeit optimierte Prüfsumme, welche nur den IP-Header (ohne Daten) schützt.
- Die Checksumme ist so ausgelegt, dass die Dekrementierung des TTL-Felds einer Inkrementierung der Checksumme entspricht. Es ist also keine Neuberechnung der Checksumme bei der Weiterleitung von Paketen notwendig.
- Es ist lediglich Fehlererkennung aber keine Korrektur möglich.

IP-Header

- Der IP-Header beinhaltet nicht nur Quell- und Ziel-Adresse.
- Die Verwendung der wichtigsten Felder wird später in diesem Kapitel anhand von Beispielen genauer erläutert werden.

Source Address

- IP-Adresse des Absenders.

IP-Header

- Der IP-Header beinhaltet nicht nur Quell- und Ziel-Adresse.
- Die Verwendung der wichtigsten Felder wird später in diesem Kapitel anhand von Beispielen genauer erläutert werden.

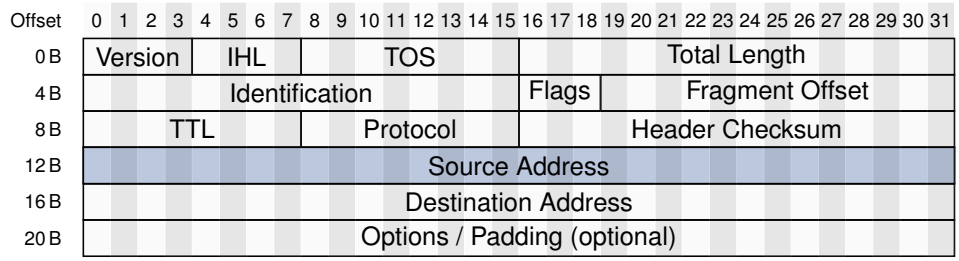


Abbildung 3.11.: IPv4-Header (minimale Länge: 20 Byte)

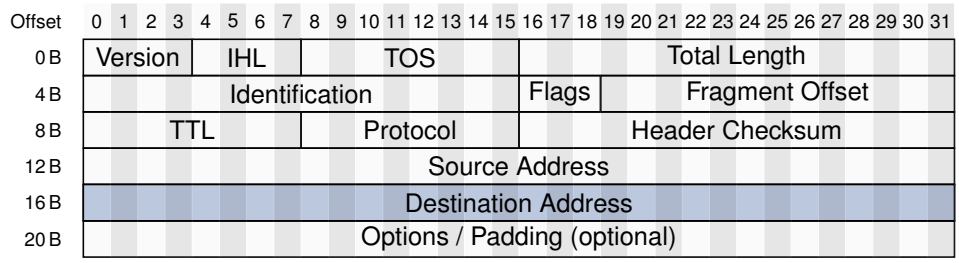


Abbildung 3.12.: IPv4-Header (minimale Länge: 20 Byte)

Destination Address

- IP-Adresse des Empfängers.

IP-Header

- Der IP-Header beinhaltet nicht nur Quell- und Ziel-Adresse.
- Die Verwendung der wichtigsten Felder wird später in diesem Kapitel anhand von Beispielen genauer erläutert werden.

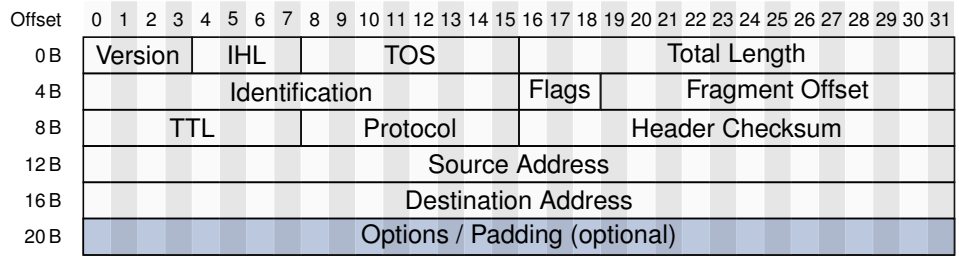


Abbildung 3.13.: IPv4-Header (minimale Länge: 20 Byte)

Options / Padding

- IP unterstützt eine Reihe von Optionen (z. B. Zeitstempel, Route Recording, ...), welche als optionale Felder an den IP-Header angefügt werden können.
- Nicht alle diese Optionen sind 4 Byte lang. Da die Länge des IP-Headers jedoch ein Vielfaches von 4 Byte betragen muss, werden kürzere Optionen ggf. durch Padding auf den nächsten gültigen Wert ergänzt.

Einschub: Host-Byte-Order und Network-Byte-Order Es gibt zwei unterschiedle Byte-Orders:

1. Big Endian: „Niederwertigstes Byte steht an höchstwertigster Adresse“
   Intuitve Schreibweise entspricht der Reihenfolge im Speicher, z. B. die Dezimalzahl 256 in hexadezimaler Schreibweise als 0x0100.

2. Little Endian: „Niederwertigstes Byte steht an niederwertigster Adresse“
   Kontraintuitiv, da die Daten im Speicher „verkehrt herum“ abgelegt werden, z. B. die Dezimalzahl 256 in hexadezimaler Schreibweise als 0x0001.

x86-kompatible Computer verwenden intern Little Endian. Bei der Kommunikation mit anderen Computern stellt das aber möglicherweise ein Problem dar. Deswegen: Konvertierung in Network Byte Order.

Network Byte Order:

Vor dem Versenden von Daten müssen Daten aus der Host Byte Order (Little Endian bei x86) in Network Byte Order (Big Endian) konvertiert werden. Analoges gilt beim Empfang von Daten.

- Dies betrifft z. B. die Felder Total Length, Identification, Flags+Fragment Offset und Header Checksum aus dem IP-Header.

- Nicht betroffen sind 1 Byte lange Felder. Quell- und Zieladresse liegen gewöhnlich als Array von 1 Byte langen Werten vor, weswegen auch diese kein Problem darstellen.

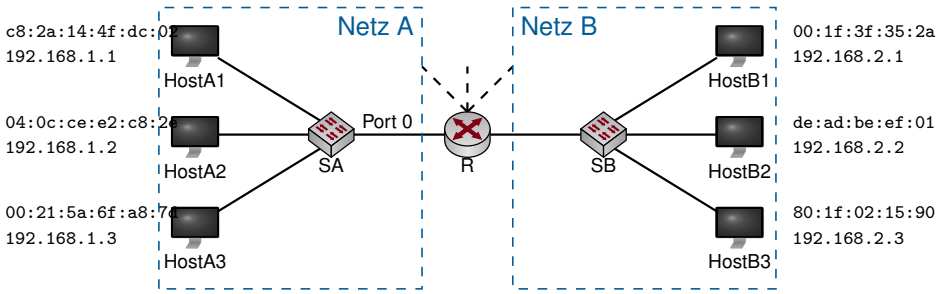
Die Konvertierung zwischen beiden Formaten erfolgt für 16 Bit lange Werte in C beispielsweise durch die Funktionen `htons()` und `ntohs()`:

- `htons(0x0001) = 0x0100` „Host to Network Short“

- `ntohs(0x0100) = 0x0001` „Network to Host Short“

Für 32 Bit lange Werte gibt es analog `htonl()` und `ntohl()` („l“ für Long).

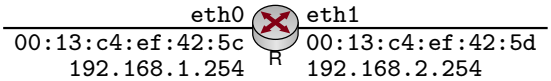
Zurück zu unserem Beispiel:

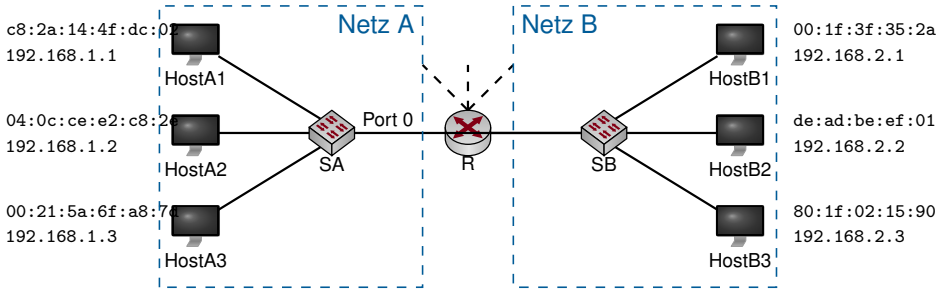


- Innerhalb von Netz A erreichen sich alle Hosts weiterhin direkt über SA.

- Will HostA1 nun ein Paket an HostB2 schicken, so geht das nicht mehr direkt:
  - Die Weiterleitung von A nach B erfolgt über R **auf Basis von IP-Adressen**
  - HostA1 muss also dafür sorgen, dass R das Paket erhält
  - Dazu trägt HostA1 als Ziel-MAC-Adresse die Adresse des lokalen Interfaces von R ein
  - Als Ziel-IP-Adresse wird die IP-Adresse von HostB2 verwendet

⇒ R muss adressierbar sein und verfügt daher auf jedem Interface über eine eigene MAC-Adresse und (wie wir gleich sehen werden) auch über eine eigene IP-Adresse:





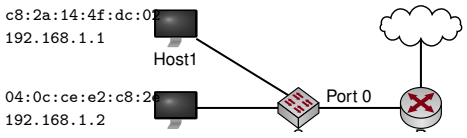
Offene Probleme:

- Angenommen der Sender kennt nur die IP-Adresse des Ziels. Wie kann die MAC-Adresse des Next-Hops bestimmt werden? (Adressauflösung)
- Woher weiß HostA1, dass er Netz B über R erreichen kann? (Routing Table)
- Angenommen das Ziel eines Pakets befindet sich nicht in einem direkt an R angeschlossenen Netz. Woher kennt R die richtige Richtung? (Routing Table)
- Wie wird diese „Routing Table“ erzeugt? (statisches Routing, Routing Protokolle)
- Was ist, wenn R mehrere Wege zu einem Ziel kennt? (Longest Prefix Matching; Border Gateway Protocol Path Selection)
- Wie funktioniert die Unterteilung einer IP-Adresse in Netz- und Hostanteil? (Classful Routing, Classless Routing, Subnetting)
- Woher kennt der Sender überhaupt die IP des Ziels? (DNS → Schicht 7)

3.3.2. Adressauflösung

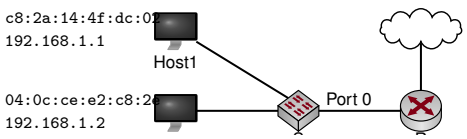
Adressauflösung [7]

- Host1 will eine Nachricht an Host2 senden
- Die IP-Adresse von Host2 (192.168.1.2) sei ihm bereits bekannt
- Wie erhält Host1 die zugehörige MAC-Adresse?



Adressauflösung [7]

- Host1 will eine Nachricht an Host2 senden
- Die IP-Adresse von Host2 (192.168.1.2) sei ihm bereits bekannt
- Wie erhält Host1 die zugehörige MAC-Adresse?



Address Resolution Protocol (ARP)

1. Host1 sendet einen ARP Request: „Who has 192.168.1.2? Tell 192.168.1.1“
2. Host2 antwort mit einem ARP Reply: „192.168.1.2 is at `04:0c:ce:e2:c8:2e`“

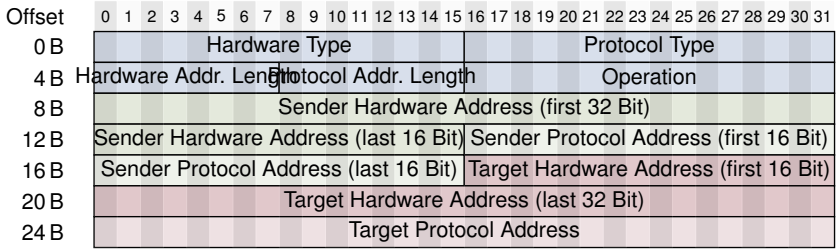
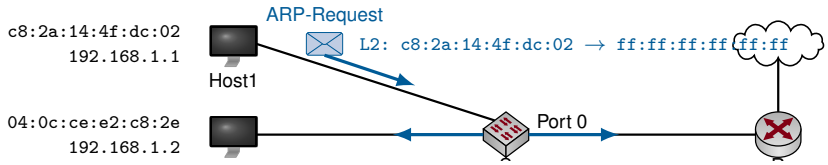
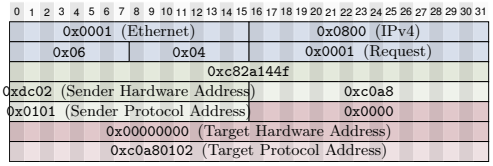


Abbildung 3.14.: ARP-Paket für IPv4 über Ethernet

Beispiel:



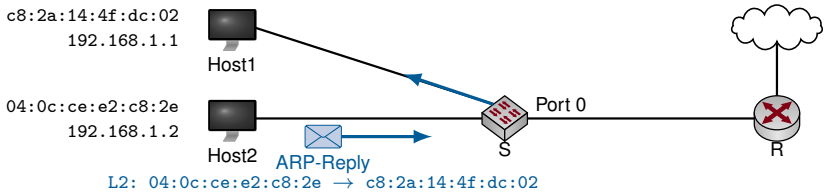
Hinweis: `L2: xx:xx:xx:xx:xx:xx → yy:yy:yy:yy:yy:yy` stellt Absender- und Ziel-MAC auf Schicht 2 dar.



(a) ARP Request

- Der ARP-Request wird an die MAC-Broadcastadresse `ff:ff:ff:ff:ff:ff` geschickt, weswegen der Switch S den Rahmen an alle angeschlossenen Hosts weiterleitet.

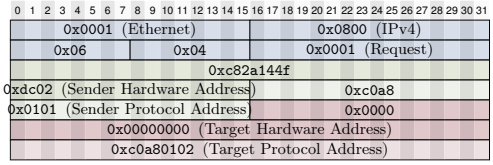
Beispiel:



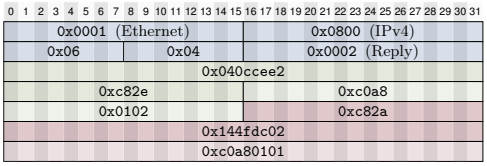
Hinweis: `L2: xx:xx:xx:xx:xx:xx → yy:yy:yy:yy:yy:yy` stellt Absender- und Ziel-MAC auf Schicht 2 dar.

- Der ARP-Request wird an die MAC-Broadcastadresse `ff:ff:ff:ff:ff:ff` geschickt, weswegen der Switch S den Rahmen an alle angeschlossenen Hosts weiterleitet.
- Der ARP-Reply wird als MAC-Unicast versendet (adressiert an Host1).
- Die Rollen Sender / Target sind zwischen Request und Reply vertauscht (vgl. Inhalte der grünen und roten Felder).

Was ist nun, wenn das Ziel nicht im selben Netz liegt (z. B. HostA1 an HostB2)?

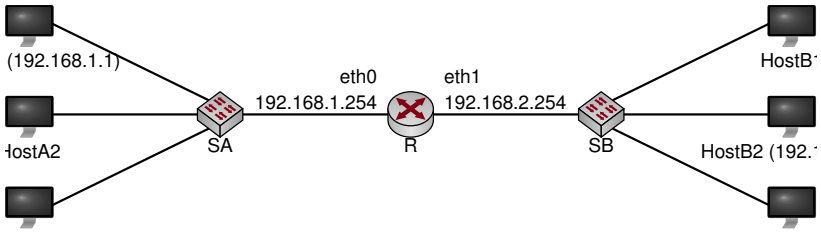


(a) ARP Request



(b) ARP Reply

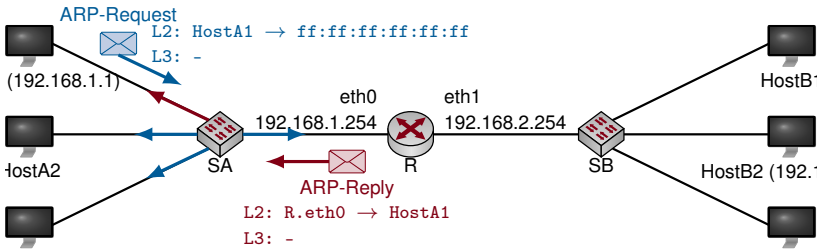
- Jeder Host sollte ein Default Gateway kennen, an das er alle Pakete schickt, deren Zieladressen nicht im eigenen Netz liegen.
- Die Zugehörigkeit einer Adresse zum jeweiligen Netz erkennt ein Host durch Vergleich der eigenen Adresse mit der Zieladdresse.
- Im Moment gehen wir noch davon aus, dass die ersten 3 Oktette einer IP-Adresse das Netz identifizieren
  ⇒ 192.168.1.1 und 192.168.2.2 liegen in unterschiedlichen Netzen.



1. HostA1 erkennt, dass 192.168.2.2 nicht im eigenen Netz liegt. Sein Default-Gateway ist 192.168.1.254.

Was ist nun, wenn das Ziel nicht im selben Netz liegt (z. B. HostA1 an HostB2)?

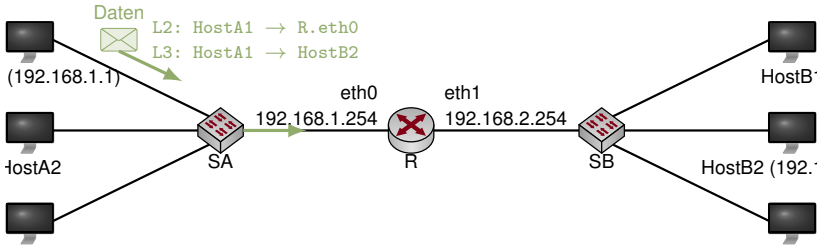
- Jeder Host sollte ein Default Gateway kennen, an das er alle Pakete schickt, deren Zieladressen nicht im eigenen Netz liegen.
- Die Zugehörigkeit einer Adresse zum jeweiligen Netz erkennt ein Host durch Vergleich der eigenen Adresse mit der Zieladdresse.
- Im Moment gehen wir noch davon aus, dass die ersten 3 Oktette einer IP-Adresse das Netz identifizieren
  ⇒ 192.168.1.1 und 192.168.2.2 liegen in unterschiedlichen Netzen.



1. HostA1 erkennt, dass 192.168.2.2 nicht im eigenen Netz liegt. Sein Default-Gateway ist 192.168.1.254.
2. HostA1 löst die MAC-Adresse zu 192.168.1.254 auf.

Was ist nun, wenn das Ziel nicht im selben Netz liegt (z. B. HostA1 an HostB2)?

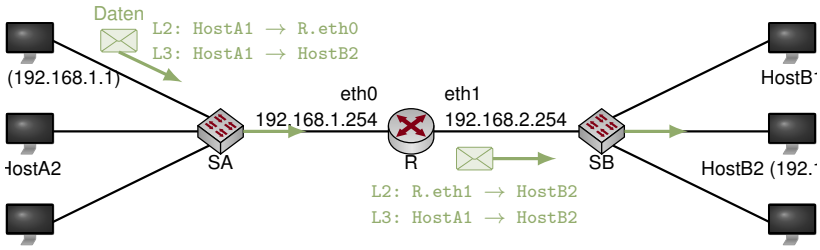
- Jeder Host sollte ein Default Gateway kennen, an das er alle Pakete schickt, deren Zieladressen nicht im eigenen Netz liegen.
- Die Zugehörigkeit einer Adresse zum jeweiligen Netz erkennt ein Host durch Vergleich der eigenen Adresse mit der Zieladdresse.
- Im Moment gehen wir noch davon aus, dass die ersten 3 Oktette einer IP-Adresse das Netz identifizieren
  ⇒ 192.168.1.1 und 192.168.2.2 liegen in unterschiedlichen Netzen.



1. HostA1 erkennt, dass 192.168.2.2 nicht im eigenen Netz liegt. Sein Default-Gateway ist 192.168.1.254.
2. HostA1 löst die MAC-Adresse zu 192.168.1.254 auf.
3. HostA1 sendet das Datenpaket an R: Dabei adressiert er R mittels der eben bestimmten MAC-Adresse (Schicht 2). Als Ziel-IP-Adresse (Schicht 3) verwendet er die IP-Adresse von HostB2.

Was ist nun, wenn das Ziel nicht im selben Netz liegt (z. B. HostA1 an HostB2)?

- Jeder Host sollte ein Default Gateway kennen, an das er alle Pakete schickt, deren Zieladressen nicht im eigenen Netz liegen.
- Die Zugehörigkeit einer Adresse zum jeweiligen Netz erkennt ein Host durch Vergleich der eigenen Adresse mit der Zieladdresse.
- Im Moment gehen wir noch davon aus, dass die ersten 3 Oktette einer IP-Adresse das Netz identifizieren
  ⇒ 192.168.1.1 und 192.168.2.2 liegen in unterschiedlichen Netzen.



1. HostA1 erkennt, dass 192.168.2.2 nicht im eigenen Netz liegt. Sein Default-Gateway ist 192.168.1.254.
2. HostA1 löst die MAC-Adresse zu 192.168.1.254 auf.
3. HostA1 sendet das Datenpaket an R: Dabei adressiert er R mittels der eben bestimmten MAC-Adresse (Schicht 2). Als Ziel-IP-Adresse (Schicht 3) verwendet er die IP-Adresse von HostB2.
4. R akzeptiert das Paket, bestimmt das ausgehende Interface und leitet das Paket weiter an HostB2. Dabei adressiert R wiederum HostB2 anhand seiner MAC-Adresse (erfordert ggf. einen weiteren ARP-Schritt).

Merke::

- MAC-Adressen dienen zur Adressierung innerhalb eines (Direktverbindungs-)Netzes und werden beim Routing verändert.
- IP-Adressen dienen der End-zu-End-Adressierung zwischen mehreren (Direktverbindungs-)Netzen und werden beim Routing nicht verändert.

Anmerkungen

- Das Ergebnis einer Adressauflösung wird i. d. R. im ARP-Cache eines Hosts zwischengespeichert, um nicht bei jedem zu versendenden Paket erneut eine Adressauflösung durchführen zu müssen.

- Die Einträge im ARP-Cache altern und werden nach einer vom Betriebssystem festgelegten Zeit invalidiert (5-10 Minuten).
- Den Inhalt des ARP-Caches kann man sich unter Linux, OS X und Windows mittels des Befehls `arp -a` anzeigen lassen.
- ARP-Replies können auch als MAC-Broadcast verschickt werden, so dass alle Hosts innerhalb einer Broadcast-Domain den Reply erhalten. Abhängig vom Betriebssystem werden derartige „unaufgeforderten ARP-Replies“ (engl. unsolicited ARP replies) häufig ebenfalls im ARP-Cache gespeichert.

Fragen: Was würde passieren, wenn ...

- zwei Hosts innerhalb derselben Broadcast-Domain identische MAC-Adressen aber unterschiedliche IP-Adressen haben?
- ein Host absichtlich auf ARP-Requests antwortet, die nicht an ihn gerichtet waren?
- ein Host unsinnige ARP-Replies via MAC-Broadcasts verschickt?

3.3.3. Internet Control Message Protocol (ICMP)

Internet Control Message Protocol (ICMP) [8] Das IP-Protokoll unterstützt das Senden von Paketen an Ziel-Hosts. Dabei kann es zu Fehlern kommen, z. B.

- ein Paket gerät in eine Routing-Schleife,
- ein Router kennt keinen Weg zum Zielnetz,
- der letzte Router zum Ziel kann die MAC-Adresse des Empfängers nicht auflösen ...

Das Internet Control Message Protocol (ICMP) dient dazu,

- in derartigen Fällen den Absender über das Problem zu benachrichtigen und
- stellt zusätzlich Möglichkeiten bereit, um z. B.
  - die Erreichbarkeit von Hosts zu prüfen („Ping“) oder
  - Pakete umzuleiten (Redirect).

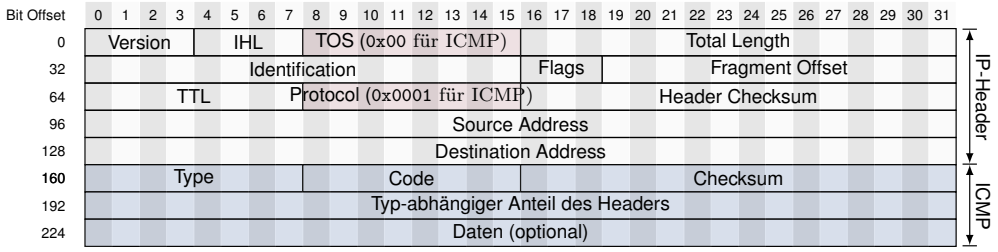
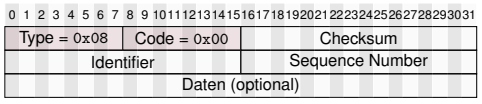
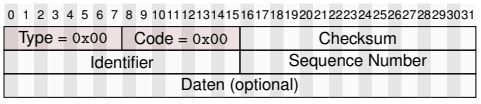


Abbildung 3.15.: Allgemeiner ICMP-Header mit vorangestelltem IP-Header

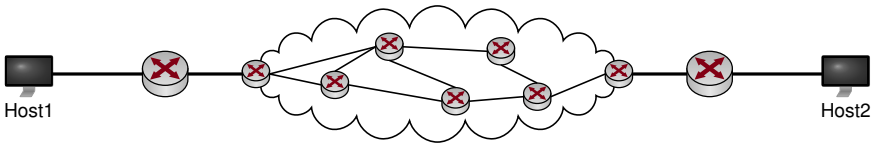
ICMP Echo Request / Echo Reply



(a) ICMP Echo Request



(b) ICMP Echo Reply

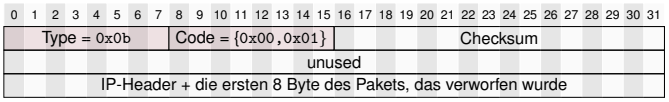


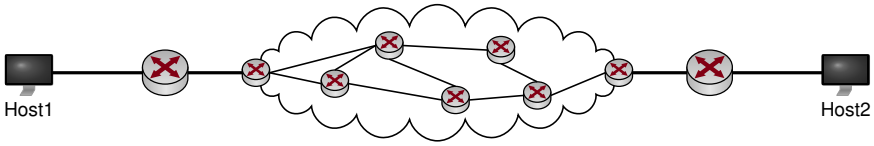
„Ping“ von Host1 nach Host2:

- Host1 wählt einen zufälligen Identifier (16 Bit Wert), die Sequenznummer wird für jeden gesendeten Echo-Request um eins inkrementiert.
- Der Echo Request wird von Routern wie jedes IP-Paket weitergeleitet.
- Erhält Host2 den Echo Request, so antwortet er mit einem Echo Reply. Dabei werden Identifier, Sequenznummer und Daten aus dem Request kopiert und zurückgeschickt.
- Sollte die Weiterleitung zu Host2 fehlschlagen, so wird eine ICMP-Nachricht mit dem entsprechenden Fehlercode an Host1 zurückgeschickt.

Frage: Wozu dient der Identifier?

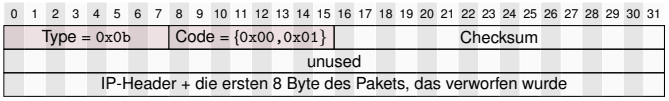
ICMP Time Exceeded

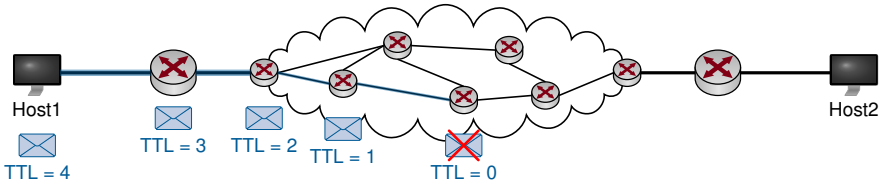




- Der IP-Header besitzt ein TTL-Feld, welches bei der Weiterleitung eines Pakets durch den jeweiligen Router um eins dekrementiert wird.

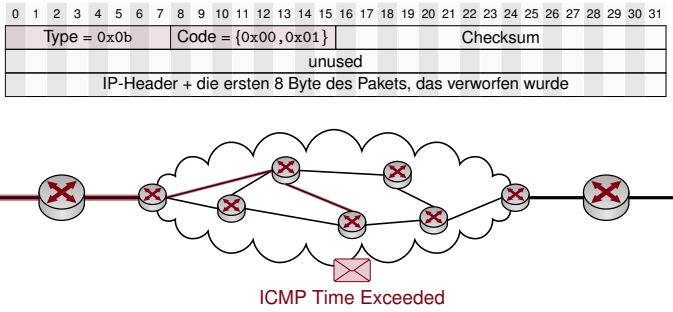
ICMP Time Exceeded





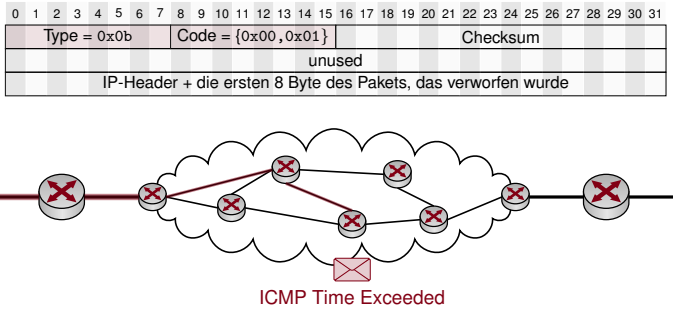
- Der IP-Header besitzt ein TTL-Feld, welches bei der Weiterleitung eines Pakets durch den jeweiligen Router um eins dekrementiert wird.
- Erreicht es den Wert 0, so wird das betreffende Paket verworfen.

ICMP Time Exceeded



- Der IP-Header besitzt ein TTL-Feld, welches bei der Weiterleitung eines Pakets durch den jeweiligen Router um eins dekrementiert wird.
- Erreicht es den Wert 0, so wird das betreffende Paket verworfen.
- Der Router generiert ein ICMP Time Exceeded und schickt es an den Absender des verworfenen Pakets zurück.

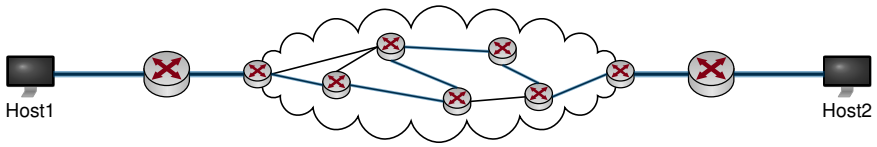
ICMP Time Exceeded



- Der IP-Header besitzt ein TTL-Feld, welches bei der Weiterleitung eines Pakets durch den jeweiligen Router um eins dekrementiert wird.
- Erreicht es den Wert 0, so wird das betreffende Paket verworfen.
- Der Router generiert ein ICMP Time Exceeded und schickt es an den Absender des verworfenen Pakets zurück.

Da der Header und die ersten 8 Byte des verworfenen Pakets an den Absender zurückgeschickt werden, kann dieser genau bestimmen, welches Paket verworfen wurde. **Frage**: Welche Informationen sind in diesen ersten 8 Byte enthalten, wenn das verworfene Paket ein ICMP Echo Request war?

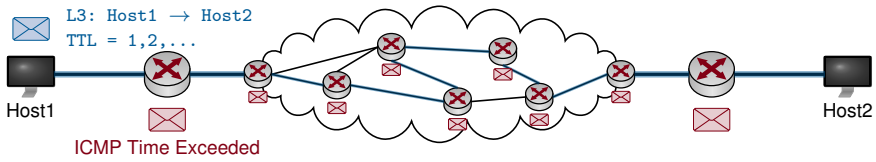
ICMP Traceroute



- Obwohl Pakete zwischen Host1 und Host2 (in Hin- und Rückrichtung) jeweils unterschiedliche Wege nehmen können, werden in der Praxis meist nur einer oder sehr wenige genutzt.

- Welchen Pfad nehmen Pakete von Host1 nach Host2?

ICMP Traceroute

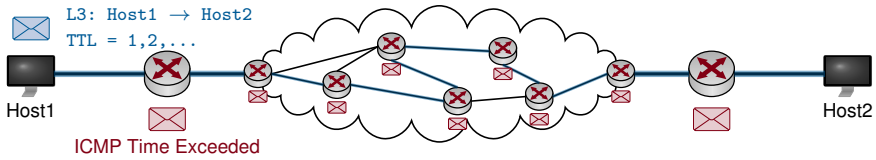


- Obwohl Pakete zwischen Host1 und Host2 (in Hin- und Rückrichtung) jeweils unterschiedliche Wege nehmen können, werden in der Praxis meist nur einer oder sehr wenige genutzt.
- Welchen Pfad nehmen Pakete von Host1 nach Host2?

Traceroute: Host1 sendet z. B. ICMP Echo Requests an Host2

- wobei das TTL-Feld zu Beginn auf 1 gesetzt wird und
- danach schrittweise um jeweils 1 erhöht wird.

⇒ Router entlang des Pfads von Host1 nach Host2 werden schrittweise die Pakete verwerfen und jeweils ein TTL Exceeded an Host1 zurücksenden. Anhand der IP-Quelladresse dieser Fehlernachrichten kann Host1 den Pfad hin zu Host2 nachvollziehen. **ICMP Traceroute**



- Obwohl Pakete zwischen Host1 und Host2 (in Hin- und Rückrichtung) jeweils unterschiedliche Wege nehmen können, werden in der Praxis meist nur einer oder sehr wenige genutzt.
- Welchen Pfad nehmen Pakete von Host1 nach Host2?

Traceroute: Host1 sendet z. B. ICMP Echo Requests an Host2

- wobei das TTL-Feld zu Beginn auf 1 gesetzt wird und
- danach schrittweise um jeweils 1 erhöht wird.

⇒ Router entlang des Pfads von Host1 nach Host2 werden schrittweise die Pakete verwerfen und jeweils ein TTL Exceeded an Host1 zurücksenden. Anhand der IP-Quelladresse dieser Fehlernachrichten kann Host1 den Pfad hin zu Host2 nachvollziehen.

⇒ 2. Programmieraufgabe: „Implementiere Traceroute“

Beispiel:

```
slacky: moepi$ traceroute -n www.net.in.tum.de
traceroute to www.net.in.tum.de (131.159.15.49), 64 hops max, 52 byte packets
1  192.168.1.1  2.570 ms  1.808 ms  2.396 ms
2  82.135.16.28  15.036 ms  14.359 ms  13.760 ms
3  212.18.7.189  14.118 ms  13.801 ms  13.845 ms
4  82.135.16.102  20.062 ms  20.137 ms  20.251 ms
5  80.81.192.222  22.707 ms  23.049 ms  23.215 ms
6  188.1.144.142  31.068 ms  36.542 ms  30.823 ms
7  188.1.37.90  30.815 ms  30.671 ms  30.808 ms
8  129.187.0.150  30.272 ms  30.602 ms  30.845 ms
9  131.159.252.1  30.885 ms  30.551 ms  30.992 ms
10  131.159.252.150  30.886 ms  30.955 ms  30.621 ms
11  131.159.252.150  30.578 ms  30.699 ms *
```

Fragen / Probleme:

- Ein Router hat mehrere IP-Adressen. Welche wählt er als Absender-Adresse der Fehlerbenachrichtigungen? Wählt er immer dieselbe Adresse?
- Was ist, wenn es tatsächlich mehrere gleichzeitig genutzte Pfade oder Pfadabschnitte gibt?
- Müssen Router überhaupt Fehlerbenachrichtigungen versenden?
- Angenommen der Pfade von $A \rightarrow B$ ist symmetrisch. Warum werden sich die Ausgaben von Traceroute dennoch unterscheiden?

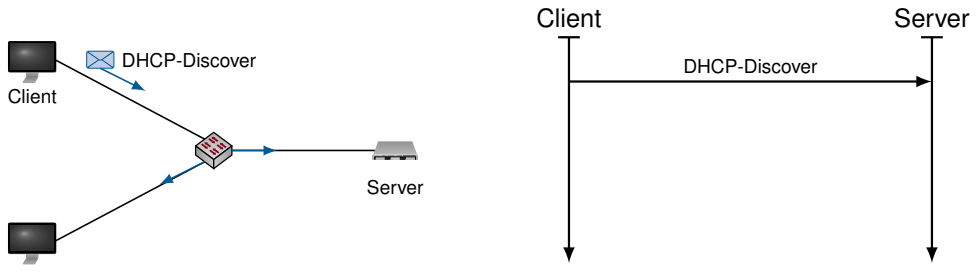
3.3.4. Dynamic Host Configuration Protocol (DHCP)

Dynamic Host Configuration Protocol (DHCP) [8] Woher bekommen Hosts eigentlich ihre IP-Adresse?

- Statische Konfiguration von Hand
- Dynamisch von einem DHCP-Server zugewiesene IP-Adresse

Ablauf:

1. Client sendet DHCP-Discover (Layer 2 Broadcast)

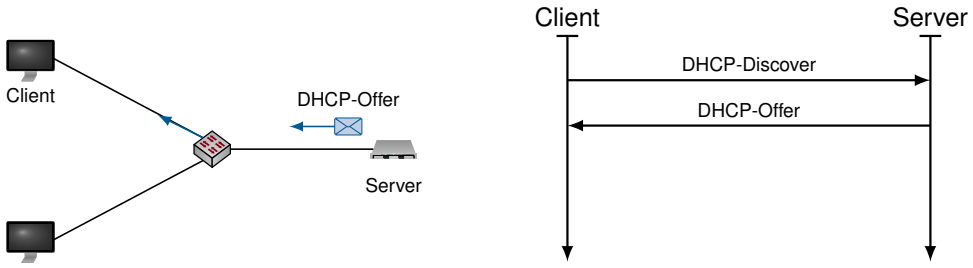


Dynamic Host Configuration Protocol (DHCP) [8] Woher bekommen Hosts eigentlich ihre IP-Adresse?

- Statische Konfiguration von Hand
- Dynamisch von einem DHCP-Server zugewiesene IP-Adresse

Ablauf:

1. Client sendet DHCP-Discover (Layer 2 Broadcast)
2. DHCP-Server antwortet mit DHCP-Offer, wodurch er dem Client eine IP-Adresse anbietet

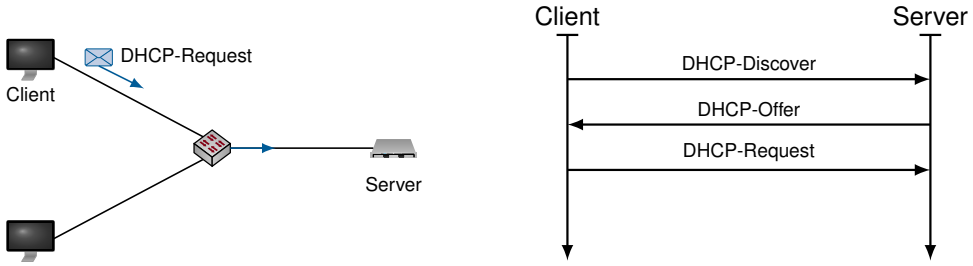


Dynamic Host Configuration Protocol (DHCP) [8] Woher bekommen Hosts eigentlich ihre IP-Adresse?

- Statische Konfiguration von Hand
- Dynamisch von einem DHCP-Server zugewiesene IP-Adresse

Ablauf:

1. Client sendet DHCP-Discover (Layer 2 Broadcast)
2. DHCP-Server antwortet mit DHCP-Offer, wodurch er dem Client eine IP-Adresse anbietet
3. Client antwortet mit DHCP-Request, wodurch er die angebotene Adresse anfordert

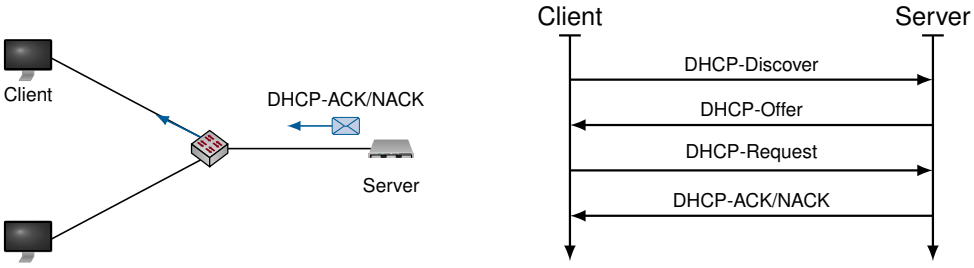


Dynamic Host Configuration Protocol (DHCP) [8] Woher bekommen Hosts eigentlich ihre IP-Adresse?

- Statische Konfiguration von Hand
- Dynamisch von einem DHCP-Server zugewiesene IP-Adresse

Ablauf:

1. Client sendet DHCP-Discover (Layer 2 Broadcast)
2. DHCP-Server antwortet mit DHCP-Offer, wodurch er dem Client eine IP-Adresse anbietet
3. Client antwortet mit DHCP-Request, wodurch er die angebotene Adresse anfordert
4. DHCP-Server antwortet mit DHCP-ACK, wodurch er die angeforderte Adresse zur Nutzung freigibt, oder mit DHCP-NACK, wodurch er die Nutzung der Adresse untersagt



Anmerkungen

- Die vom DHCP-Server zugewiesene Adresse wird auch als Lease bezeichnet und ist in ihrer Gültigkeit zeitlich begrenzt
- Clients erneuern ihr Lease in regelmäßigen Abständen beim DCHP-Server.
- Gerade in kleineren (privaten) Netzwerken übernimmt häufig ein Router die Rolle des DHCP-Servers.
- Zusammen mit der IP-Adresse und Subnetzmaske können DHCP-Server viele weitere Optionen ausliefern:
  - DNS-Server zur Namensauflösung (→ Kapitel 6)
  - Hostname und Domänen-Suffix (→ Kapitel 6)
  - Statische Routen, insbesondere einen Default-Gateway
  - Maximum Transmission Unit
  - NTP-Server zur Zeitsynchronisation
  - ...
- Aus Redundanzgründen werden manchmal mehrere DHCP-Server pro Netzwerk verwendet, wobei
  - sich die Adressbereiche der beiden Server nicht überschneiden dürfen oder
  - zusätzliche Mechanismen zur Synchronisation der Adresspools notwendig sind.
- Ein versehentlich ins Netzwerk eingebrachter DHCP-Server (z. B. durch einen achtlos angeschlossenen Router oder Access Point) kann beträchtliche Auswirkungen auf das Netzwerk haben und ist häufig schwer zu finden:
  - Im Netz eines Lehrstuhls gab es einst einen solchen Rogue-DHCP-Server,
  - der trotz intensiver Suche nicht gefunden werden konnte.
  - Über die MAC-Adressen der vom Server stammdenden DHCP-Nachrichten stellte sich am Ende heraus, dass es sich um ein **Nokia-Smartphone** im WLAN handelte!
  - Die Anzahl der Verdächtigen hatte sich damit stark eingeschränkt (;

3.3.5. Adressklassen (Classful Routing)

Adressklassen (Classful Routing) Zu Beginn des Kapitels hatten wir in einem Beispiel beschrieben, dass

- die ersten drei Oktette einer IP-Adresse das Netz identifizieren und
- das vierte Oktett Hosts innerhalb eines Netzes adressiert.

Diese Aufteilung war lediglich ein Beispiel, entsprechend der Adress-Klasse C. Historisch ist der IP-Adresseraum in folgende fünf Klassen unterteilt:

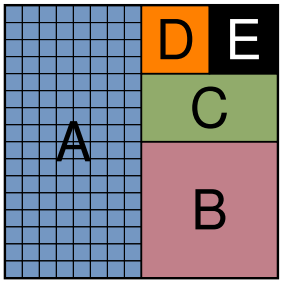
Klasse	1. Oktett	Erste Adresse	Letzte Adresse	Netz-/Hostanteil	Anzahl Netze	Adressen pro Netz
A	`0xxxxxxx`	0.0.0.0	127.255.255.255	N.H.H.H	$2^7 = 128$	$2^{24} = 16777216$
B	`10xxxxxx`	128.0.0.0	191.255.255.255	N.N.H.H	$2^{14} = 16384$	$2^{16} = 65536$
C	`110xxxxx`	192.0.0.0	223.255.255.255	N.N.N.H	$2^{21} = 2097152$	$2^8 = 256$
D	`1110xxxx`	224.0.0.0	239.255.255.255	Reserviert für Multicast		
E	`1111xxxx`	240.0.0.0	255.255.255.255	Reserviert		

Grafische Darstellung:

- IP-Adressraum als Quadrat
- Farbige Flächen markieren die fünf Adressklassen

- Die einzelnen Klasse-A-Netze sind als Drahtgitter angedeutet

→ Nächste Folie zeigt die tatsächliche Verteilung und Ausnutzung des Adressraums [1] (selbe Farbkodierung).



Verschwendung des Adressraums

- Als IPv4 1981 eingeführt wurde, konnte man sich nicht vorstellen, dass $\tilde{}2^{32}$ Adressen aufgeteilt in die Klassen A, B und C nicht ausreichend würden.
- Es wurden große Adresseblöcke an Firmen, Behörden und Bildungseinrichungen vergeben, z. B. ganze Klasse-A Netze an HP, IBM, AT&T, Apple, MIT, Generel Electric, US Army, ...

Folge:

- Ineffiziente Aufteilung und Nutzung des Adressraums
- Große Netze mit internen Routern ⇒ weitere Unterteilung in Subnetze notwendig

Situation heute:

- Vergabe des letzten IPv4 Adressblocks am 3.2.2011 durch die IANA[1] an eine der fünf Regional Internet Registries (RIRs), das APNIC[2]
- IPv4-Adressraum praktisch aufgebraucht
- Immernoch schleppende Einführung von IPv6

3.3.6. Subnetting (Classless Routing)

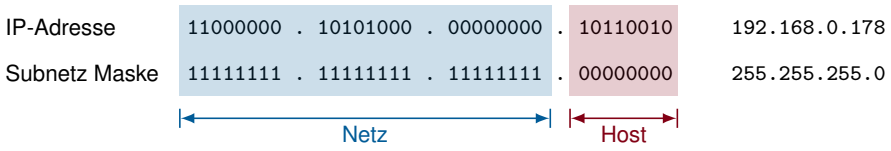
Subnetting (Classless Routing) Bereits 1993 wurde mit CIDR[3] ein Verfahren zur Unterteilung von IP-Netzen eingeführt:

- Zusätzlich zur IP-Adresse erhält ein Interface eine ebenfalls 32 Bit lange Subnetzmaske
- Die Subnetzmaske unterteilt die IP-Adresse in einen Netzanteil und einen Hostanteil
- Eine logische 1 in der Subnetzmaske bedeutet Netzanteil, eine logische 0 Hostanteil
- UND-Verknüpfung von IP-Adresse und Subnetzmaske ergibt die Netzadresse

[1] Internet Assigned Numbers Authority
[2] Asia-Pacific Network Information Center (APNIC)
[3] Classless Inter-Domain Routing

- Die übliche Klassenzugehörigkeit hat damit nur noch im Sprachgebrauch eine Bedeutung

Beispiel 1:

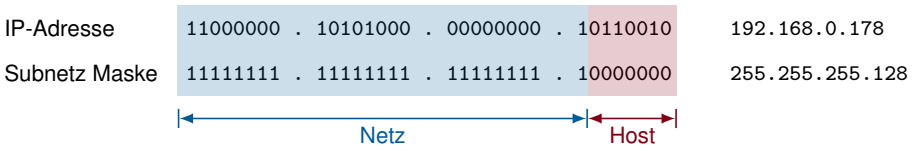


- 24 Bit Netzanteil, 8 Bit Hostanteil $\Rightarrow 2^8 = 256$ Adressen
- Netzadresse: 192.168.0.0
- Broadcastadresse: 192.168.0.255
- Nutzbare Adressen für Hosts: $2^8 - 2 = 254$

Subnetting (Classless Routing) Bereits 1993 wurde mit CIDR[4] ein Verfahren zur Unterteilung von IP-Netzen eingeführt:

- Zusätzlich zur IP-Adresse erhält ein Interface eine ebenfalls 32 Bit lange Subnetzmaske
- Die Subnetzmaske unterteilt die IP-Adresse in einen Netzanteil und einen Hostanteil
- Eine logische 1 in der Subnetzmaske bedeutet Netzanteil, eine logische 0 Hostanteil
- UND-Verknüpfung von IP-Adresse und Subnetzmaske ergibt die Netzadresse
- Die übliche Klassenzugehörigkeit hat damit nur noch im Sprachgebrauch eine Bedeutung

Beispiel 2:



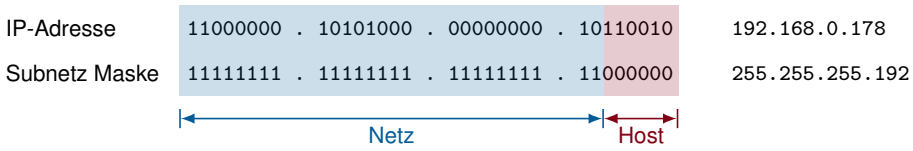
- 25 Bit Netzanteil, 7 Bit Hostanteil $\Rightarrow 2^7 = 128$ Adressen
- Netzadresse: 192.168.0.128
- Broadcastadresse: 192.168.0.255
- Nutzbare Adressen für Hosts: $2^7 - 2 = 126$

Subnetting (Classless Routing) Bereits 1993 wurde mit CIDR[5] ein Verfahren zur Unterteilung von IP-Netzen eingeführt:

- Zusätzlich zur IP-Adresse erhält ein Interface eine ebenfalls 32 Bit lange Subnetzmaske
- Die Subnetzmaske unterteilt die IP-Adresse in einen Netzanteil und einen Hostanteil
- Eine logische 1 in der Subnetzmaske bedeutet Netzanteil, eine logische 0 Hostanteil
- UND-Verknüpfung von IP-Adresse und Subnetzmaske ergibt die Netzadresse
- Die übliche Klassenzugehörigkeit hat damit nur noch im Sprachgebrauch eine Bedeutung

Beispiel 3:

[4] Classless Inter-Domain Routing
[5] Classless Inter-Domain Routing

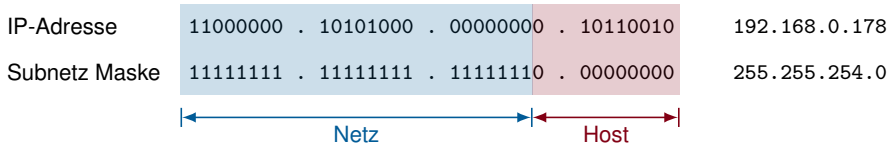


- 26 Bit Netzanteil, 6 Bit Hostanteil $\Rightarrow 2^6 = 64$ Adressen
- Netzadresse: 192.168.0.128
- Broadcastadresse: 192.168.0.191
- Nutzbare Adressen für Hosts: $2^6 - 2 = 62$

Supernetting Dasselbe Prinzip funktioniert auch in die andere Richtung:

- Zusammenfassung mehrerer zusammenhängender kleinerer Netze zu einem größeren Netz
- Wird häufig von Routern angewendet, um die Anzahl der Einträge in der Routingtabelle zu reduzieren

Beispiel:



- 23 Bit Netzanteil, 9 Bit Hostanteil $\Rightarrow 2^9 = 512$ Adressen
- Netzadresse: 192.168.0.0
- Broadcastadresse: 192.168.1.255
- Nutzbare Adressen für Hosts: $2^9 - 2 = 510$

Präfixschreibweise: Anstelle die Subnetzmaske auszuschreiben, wird häufig nur die Länge des Netzanteils (Anzahl führender Einsen in der Subnetzmaske) angegeben, z. B. 192.168.0.0/23.

Frage: Können die beiden Netze 192.168.1.0/24 und 192.168.2.0/24 zu einem größeren Netz zusammengefasst werden? **Frage:** Können die beiden Netze 192.168.1.0/24 und 192.168.2.0/24 zu einem größeren Netz zusammengefasst werden? **Antwort:** Nein, denn es gibt keine passende Subnetzmaske:

- 192.168.0.0/23 enthält die Netze 192.168.{0,1}.0/24
- 192.168.2.0/23 enthält die Netze 192.168.{2,3}.0/24
- 192.168.1.0/23 ist keine gültige Netzadresse, denn UND-Verknüpfung der Adresse 192.168.1.0 mit der Subnetzmaske 255.255.254.0 ergibt 192.168.0.0 als Netzadresse! $\Rightarrow$ 192.168.1.0 ist eine Hostadresse im Netz 192.168.0.0/23.

Frage: Können die beiden Netze 192.168.1.0/24 und 192.168.2.0/24 zu einem größeren Netz zusammengefasst werden? **Antwort:** Nein, denn es gibt keine passende Subnetzmaske:

- 192.168.0.0/23 enthält die Netze 192.168.{0,1}.0/24
- 192.168.2.0/23 enthält die Netze 192.168.{2,3}.0/24

- 192.168.1.0/23 ist keine gültige Netzadresse, denn UND-Verknüpfung der Adresse 192.168.1.0 mit der Subnetzmaske 255.255.254.0 ergibt 192.168.0.0 als Netzadresse!
  ⇒ 192.168.1.0 ist eine Hostadresse im Netz 192.168.0.0/23.

Frage: Können die beiden Netze 192.168.0.0/24 und 192.168.2.0/24 zu einem größeren Netz zusammengefasst werden? **Frage:** Können die beiden Netze 192.168.1.0/24 und 192.168.2.0/24 zu einem größeren Netz zusammengefasst werden? **Antwort:** Nein, denn es gibt keine passende Subnetzmaske:

- 192.168.0.0/23 enthält die Netze 192.168.{0,1}.0/24
- 192.168.2.0/23 enthält die Netze 192.168.{2,3}.0/24
- 192.168.1.0/23 ist keine gültige Netzadresse, denn UND-Verknüpfung der Adresse 192.168.1.0 mit der Subnetzmaske 255.255.254.0 ergibt 192.168.0.0 als Netzadresse!
  ⇒ 192.168.1.0 ist eine Hostadresse im Netz 192.168.0.0/23.

Frage: Können die beiden Netze 192.168.0.0/24 und 192.168.2.0/24 zu einem größeren Netz zusammengefasst werden? **Antwort:** Nein, denn die beiden Netze sind nicht benachbart:

- Das Netz 192.168.0.0/22 würde zwar beide Subnetze enthalten, zusätzlich aber auch die beiden Netze 192.168.{1,3}.0/24.

Frage: Können die beiden Netze 192.168.1.0/24 und 192.168.2.0/24 zu einem größeren Netz zusammengefasst werden? **Antwort:** Nein, denn es gibt keine passende Subnetzmaske:

- 192.168.0.0/23 enthält die Netze 192.168.{0,1}.0/24
- 192.168.2.0/23 enthält die Netze 192.168.{2,3}.0/24
- 192.168.1.0/23 ist keine gültige Netzadresse, denn UND-Verknüpfung der Adresse 192.168.1.0 mit der Subnetzmaske 255.255.254.0 ergibt 192.168.0.0 als Netzadresse!
  ⇒ 192.168.1.0 ist eine Hostadresse im Netz 192.168.0.0/23.

Frage: Können die beiden Netze 192.168.0.0/24 und 192.168.2.0/24 zu einem größeren Netz zusammengefasst werden? **Antwort:** Nein, denn die beiden Netze sind nicht benachbart:

- Das Netz 192.168.0.0/22 würde zwar beide Subnetze enthalten, zusätzlich aber auch die beiden Netze 192.168.{1,3}.0/24.

Frage: Können die vier Netze 192.168.{4,5,6,7}.0/24 zu einem größeren Netz zusammengefasst werden? **Frage:** Können die beiden Netze 192.168.1.0/24 und 192.168.2.0/24 zu einem größeren Netz zusammengefasst werden? **Antwort:** Nein, denn es gibt keine passende Subnetzmaske:

- 192.168.0.0/23 enthält die Netze 192.168.{0,1}.0/24
- 192.168.2.0/23 enthält die Netze 192.168.{2,3}.0/24
- 192.168.1.0/23 ist keine gültige Netzadresse, denn UND-Verknüpfung der Adresse 192.168.1.0 mit der Subnetzmaske 255.255.254.0 ergibt 192.168.0.0 als Netzadresse!
  ⇒ 192.168.1.0 ist eine Hostadresse im Netz 192.168.0.0/23.

Frage: Können die beiden Netze 192.168.0.0/24 und 192.168.2.0/24 zu einem größeren Netz zusammengefasst werden? **Antwort:** Nein, denn die beiden Netze sind nicht benachbart:

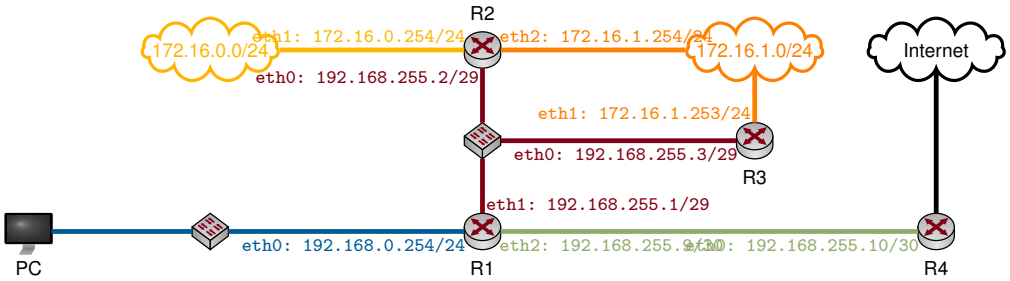
- Das Netz 192.168.0.0/22 würde zwar beide Subnetze enthalten, zusätzlich aber auch die beiden Netze 192.168.{1,3}.0/24.

Frage: Können die vier Netze 192.168.{4,5,6,7}.0/24 zu einem größeren Netz zusammengefasst werden? **Antwort:** Ja, das Netz 192.168.4.0/22 umfasst genau diese vier Netze.

3.4. Wegwahl (Routing)

3.4.0.1. Statisches Routing

Wir betrachten im Folgenden das unten abgebildete Beispielnetzwerk:

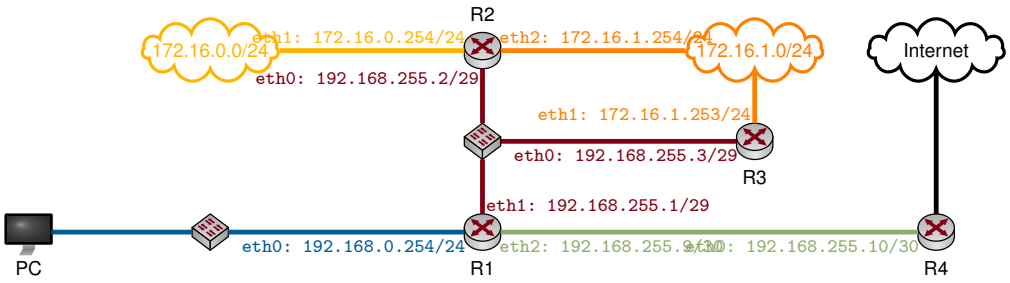


- Die Farben der Links und Interface-Adressen verdeutlichen die einzelnen Subnetze
- Das Netzwerk 192.168.255.0/29 (rot) verfügt über 6 nutzbare Hostadressen
- Das Netzwerk 192.168.255.8/30 (grün) ist ein Transportnetz mit nur 2 nutzbaren Hostadressen
- Die übrigen Netze sind /24 Netze mit jeweils 254 nutzbaren Hostadressen

Frage: Wie entscheidet R1, an welchen Next-Hop ein Paket weitergeleitet werden soll?

3.4.1. Routing Table und Longest Prefix Matching

Routing Table

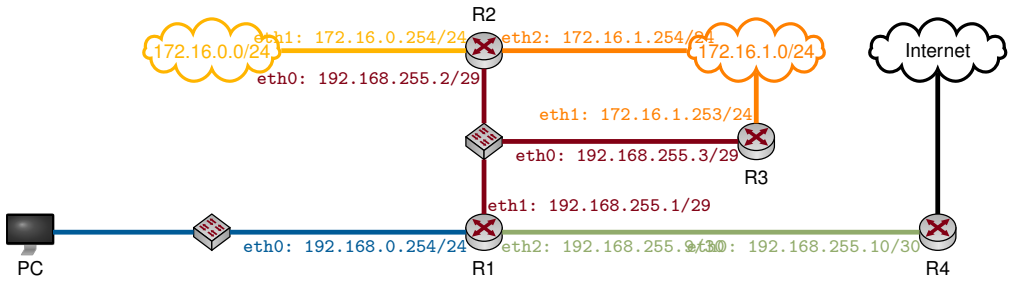


Definition: Routing Table:

In der Routing-Tabelle speichert ein Router (oder Host)

- die Netzadresse eines Ziels,
- die zugehörige Subnetzemaske (oder das Präfix),
- den zugehörigen Next-Hop (auch Gateway genannt),
- das Interface, über welches dieser Next-Hop erreichbar ist, und
- die Kosten bzw. die Metrik bis zum Ziel.

Routing Table



Beispiel: Routing-Tabelle für R1

Destination	Mask	Gateway	Metric	Iface
192.168.255.8	255.255.255.252	0.0.0.0	0	eth2
192.168.255.0	255.255.255.248	0.0.0.0	0	eth1
192.168.0.0	255.255.255.0	0.0.0.0	0	eth0
172.16.1.0	255.255.255.0	192.168.255.3	1	eth1
172.16.0.0	255.255.254.0	192.168.255.2	1	eth1
0.0.0.0	0.0.0.0	192.168.255.10	0	eth2

- Die Netze 172.16.{0.1}.0/24 wurden zusammengefasst
- Die Route 0.0.0.0 wird auch als Default Route bezeichnet
- Interessant: R1 kennt zwei (eigentlich sogar drei) Routen zum Netz 172.16.1.0/24!

Longest Prefix Matching

1. R1 berechnet das logische UND aus der Zieladresse des Pakets und den Masken („Genmasks“) in seiner Routingtabelle
2. Das Ergebnis wird mit dem Eintrag in der Spalte „Destination“ verglichen
3. Stimmt das Ergebnis damit überein, werden Gateway und zugehöriges Interface bestimmt
4. Nachdem die MAC-Adresse des Gateways ggf. via ARP aufgelöst wurde, wird das Paket mit einem neuen Ethernet-Header versehen und weitergeleitet

Beispiel: R1 erhalte ein Paket mit Zieladresse 172.16.1.23

Destination	Mask	Gateway	Metric	Iface
192.168.255.8	255.255.255.252	0.0.0.0	0	eth2
192.168.255.0	255.255.255.248	0.0.0.0	0	eth1
192.168.0.0	255.255.255.0	0.0.0.0	0	eth0
172.16.1.0	255.255.255.0	192.168.255.3	1	eth1
172.16.0.0	255.255.254.0	192.168.255.2	1	eth1
0.0.0.0	0.0.0.0	192.168.255.10	0	eth2

Longest Prefix Matching

1. R1 berechnet das logische UND aus der Zieladresse des Pakets und den Masken („Genmasks“) in seiner Routingtabelle
2. Das Ergebnis wird mit dem Eintrag in der Spalte „Destination“ verglichen
3. Stimmt das Ergebnis damit überein, werden Gateway und zugehöriges Interface bestimmt
4. Nachdem die MAC-Adresse des Gateways ggf. via ARP aufgelöst wurde, wird das Paket mit einem neuen Ethernet-Header versehen und weitergeleitet

Beispiel: R1 erhalte ein Paket mit Zieladresse 172.16.1.23

	Destination	Mask	Gateway	Metric	Iface
→	192.168.255.8	255.255.255.252	0.0.0.0	0	eth2
	192.168.255.0	255.255.255.248	0.0.0.0	0	eth1
	192.168.0.0	255.255.255.0	0.0.0.0	0	eth0
	172.16.1.0	255.255.255.0	192.168.255.3	1	eth1
	172.16.0.0	255.255.254.0	192.168.255.2	1	eth1
	0.0.0.0	0.0.0.0	192.168.255.10	0	eth2

IP-Adresse	10101100 . 00010000 . 00000001 . 00010111	172.16.1.23
Subnetz Maske	11111111 . 11111111 . 11111111 . 11111100	255.255.255.252
Netzadresse	10101100 . 00010000 . 00000001 . 00010100	172.16.1.20

⇒ kein Match, da

$$172.16.1.20 \neq 192.168.255.8$$

Longest Prefix Matching

1. R1 berechnet das logische UND aus der Zieladresse des Pakets und den Masken („Genmasks“) in seiner Routingtabelle
2. Das Ergebnis wird mit dem Eintrag in der Spalte „Destination“ verglichen
3. Stimmt das Ergebnis damit überein, werden Gateway und zugehöriges Interface bestimmt
4. Nachdem die MAC-Adresse des Gateways ggf. via ARP aufgelöst wurde, wird das Paket mit einem neuen Ethernet-Header versehen und weitergeleitet

Beispiel: R1 erhalte ein Paket mit Zieladresse 172.16.1.23

	Destination	Mask	Gateway	Metric	Iface
	192.168.255.8	255.255.255.252	0.0.0.0	0	eth2
→	192.168.255.0	255.255.255.248	0.0.0.0	0	eth1
	192.168.0.0	255.255.255.0	0.0.0.0	0	eth0
	172.16.1.0	255.255.255.0	192.168.255.3	1	eth1
	172.16.0.0	255.255.254.0	192.168.255.2	1	eth1
	0.0.0.0	0.0.0.0	192.168.255.10	0	eth2

IP-Adresse	10101100 . 00010000 . 00000001 . 00010111	172.16.1.23
Subnetz Maske	11111111 . 11111111 . 11111111 . 11111000	255.255.255.248
Netzadresse	10101100 . 00010000 . 00000001 . 00010000	172.16.1.16

⇒ kein Match, da

$$172.16.1.16 \neq 192.168.255.0$$

Longest Prefix Matching

1. R1 berechnet das logische UND aus der Zieladresse des Pakets und den Masken („Genmasks“) in seiner Routingtabelle
2. Das Ergebnis wird mit dem Eintrag in der Spalte „Destination“ verglichen
3. Stimmt das Ergebnis damit überein, werden Gateway und zugehöriges Interface bestimmt
4. Nachdem die MAC-Adresse des Gateways ggf. via ARP aufgelöst wurde, wird das Paket mit einem neuen Ethernet-Header versehen und weitergeleitet

Beispiel: R1 erhalte ein Paket mit Zieladresse 172.16.1.23

	Destination	Mask	Gateway	Metric	Iface
	192.168.255.8	255.255.255.252	0.0.0.0	0	eth2
	192.168.255.0	255.255.255.248	0.0.0.0	0	eth1
→	192.168.0.0	255.255.255.0	0.0.0.0	0	eth0
	172.16.1.0	255.255.255.0	192.168.255.3	1	eth1
	172.16.0.0	255.255.254.0	192.168.255.2	1	eth1
	0.0.0.0	0.0.0.0	192.168.255.10	0	eth2

IP-Adresse	10101100 . 00010000 . 00000001 . 00010111	172.16.1.23
Subnetz Maske	11111111 . 11111111 . 11111111 . 00000000	255.255.255.0
Netzadresse	10101100 . 00010000 . 00000001 . 00000000	172.16.1.0

⇒ kein Match, da

$$172.16.1.0 \neq 192.168.0.0$$

Longest Prefix Matching

1. R1 berechnet das logische UND aus der Zieladresse des Pakets und den Masken („Genmasks“) in seiner Routingtabelle
2. Das Ergebnis wird mit dem Eintrag in der Spalte „Destination“ verglichen
3. Stimmt das Ergebnis damit überein, werden Gateway und zugehöriges Interface bestimmt
4. Nachdem die MAC-Adresse des Gateways ggf. via ARP aufgelöst wurde, wird das Paket mit einem neuen Ethernet-Header versehen und weitergeleitet

Beispiel: R1 erhalte ein Paket mit Zieladresse 172.16.1.23

	Destination	Mask	Gateway	Metric	Iface
	192.168.255.8	255.255.255.252	0.0.0.0	0	eth2
	192.168.255.0	255.255.255.248	0.0.0.0	0	eth1
	192.168.0.0	255.255.255.0	0.0.0.0	0	eth0
→	172.16.1.0	255.255.255.0	192.168.255.3	1	eth1
	172.16.0.0	255.255.254.0	192.168.255.2	1	eth1
	0.0.0.0	0.0.0.0	192.168.255.10	0	eth2

IP-Adresse	10101100 . 00010000 . 00000001 . 00010111	172.16.1.23
Subnetz Maske	11111111 . 11111111 . 11111111 . 00000000	255.255.255.0
Netzadresse	10101100 . 00010000 . 00000001 . 00000000	172.16.1.0

⇒ Match, da 172.16.1.0 =

172.16.1.0 ⇒ Gateway ist 192.168.255.3

Definition: Longest Prefix Matching:

Die Routingtabelle wird von längeren Präfixen (spezifischeren Routen) hin zu kürzeren Präfixen (weniger spezifische Routen) durchsucht. Der erste passende Eintrag liefert das Gateway (Next-Hop) eines Pakets. Diesen Prozess bezeichnet man als Longest Prefix Matching.

Beachte:

Destination	Mask	Gateway	Metric	Iface
192.168.255.8	255.255.255.252	0.0.0.0	0	eth2
192.168.255.0	255.255.255.248	0.0.0.0	0	eth1
192.168.0.0	255.255.255.0	0.0.0.0	0	eth0
172.16.1.0	255.255.255.0	192.168.255.3	1	eth1
172.16.0.0	255.255.254.0	192.168.255.2	1	eth1
0.0.0.0	0.0.0.0	192.168.255.10	0	eth2

- Der Eintrag für 172.16.0.0/23 liefert ebenfalls einen Match, ist aber weniger spzifisch als der für 172.16.1.0/24 (1 Bit kürzeres Präfix).
- Die Default Route 0.0.0.0 liefert immer einen Match (logisches UND mit der Maske 0.0.0.0).
- Es ist nicht garantiert, dass das Gateway der Default Route („Gateway of last resort“) eine Route zum Ziel kennt (→ ICMP Destination Unreachable / Host Unreachable).
- Routen zu direkt verbundenen Netzen (also solchen, zu denen ein Router selbst gehört) können automatisch erzeugt werden. Das Gateway ist in diesem Fall der Router selbst (bzw. eine seiner IP-Adressen).
- Routen zu entfernten Netzen müssen „gelernt“ werden – entweder durch händisches Eintragen (statisches Routing) oder durch Routing Protokolle (dynamisches Routing).

3.4.2. Dynamisches Routing

Dynamisches Routing Mittels Routing Protokollen können Router miteinander kommunizieren und Routen untereinander austauschen. Routingprotokolle können nach Ihrer Funktionsweise wie folgt gruppiert werden: **Distanz-Vektor-Protokolle**

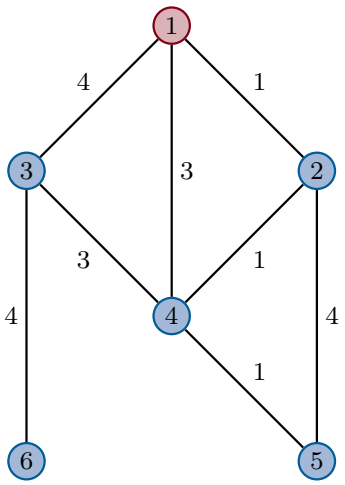
- Router kennen nur Richtung (Next Hop) und Entfernung (Kosten) zu einem Ziel (vgl. Straßenschild mit Richtungs- und Entfernungsangabe)
- Router haben keine Information über die Netzwerktopologie
- Router tauschen untereinander lediglich kumulierte Metriken aus (z. B. den Inhalt Ihrer Routingtabellen)
- Funktionsprinzip basiert auf dem Algorithmus von Bellman-Ford

Link-State-Protokolle

- Router informieren einander detailliert über den aktuellen Zustand einzelner Links
- Router verfügen über vollständige Topologieinformationen
- Häufig komplexe Nachbarschaftsbeziehungen und Update-Nachrichten
- Basierend auf den Topologieinformationen bestimmt jeder Router kürzeste Pfade, z. B. mittels Dijkstras Algorithmus

Algorithmus von Bellman-Ford

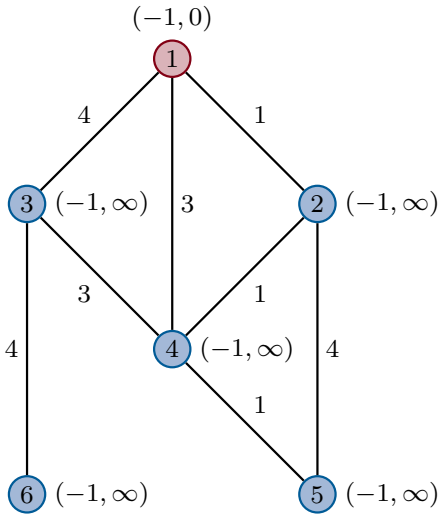
- $p[i]$: Vorgänger von Knoten i im Graphen
- $d[i]$: Distanz von der Wurzel zu Knoten i



Algorithmus von Bellman-Ford

- $p[i]$: Vorgänger von Knoten i im Graphen
- $d[i]$: Distanz von der Wurzel zu Knoten i

// Initialisierung
for $i \in \mathcal{N}$ **do**
  $p[i] = -1$
  $d[i] = \begin{cases} 0 & i = s \\ \infty & \text{sonst} \end{cases}$
end for
$T = \{s\}$ // Menge der erreichbaren Knoten

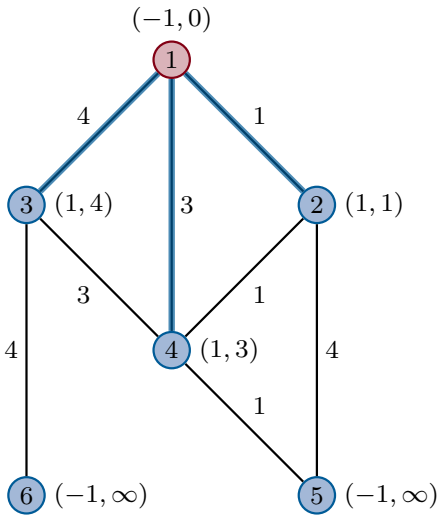


Algorithmus von Bellman-Ford

- $p[i]$: Vorgänger von Knoten i im Graphen
- $d[i]$: Distanz von der Wurzel zu Knoten i

// Initialisierung
for $i \in \mathcal{N}$ **do**
  $p[i] = -1$
  $d[i] = \begin{cases} 0 & i = s \\ \infty & \text{sonst} \end{cases}$
end for
$T = \{s\}$ // Menge der erreichbaren Knoten

// Berechnung der Pfade
while $d[j]$ changes for some $j \in \mathcal{N}$ **do**
  $S = \{\}$ // Aktualisierte Knoten
  for $i \in T$ **do**
    // Für alle Nachbarn von i:
    for $\forall j : (i, j) \in \mathcal{E}$ **do**
      if $d[i] + c_{ij} < d[j]$ **then**
        $p[j] = i$
        $d[j] = d[i] + c_{ij}$
        $S = S \cup \{j\}$
      end if
    end for
  end for
  $T = T \cup S$
end while



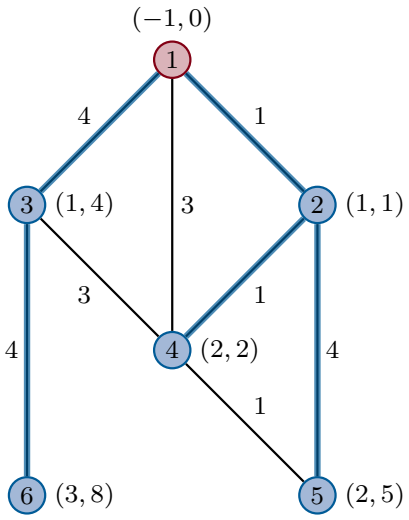
Algorithmus von Bellman-Ford

- $p[i]$: Vorgänger von Knoten i im Graphen

- $d[i]$: Distanz von der Wurzel zu Knoten i

// Initialisierung
for $i \in \mathcal{N}$ **do**
  $p[i] = -1$
  $d[i] = \begin{cases} 0 & i = s \\ \infty & \text{sonst} \end{cases}$
end for
$T = \{s\}$ // Menge der erreichbaren Knoten

// Berechnung der Pfade
while $d[j]$ changes for some $j \in \mathcal{N}$ **do**
  $S = \{\}$ // Aktualisierte Knoten
  for $i \in T$ **do**
    // Für alle Nachbarn von i:
    for $\forall j : (i, j) \in \mathcal{E}$ **do**
      if $d[i] + c_{ij} < d[j]$ **then**
        $p[j] = i$
        $d[j] = d[i] + c_{ij}$
        $S = S \cup \{j\}$
      end if
    end for
  end for
  $T = T \cup S$
end while



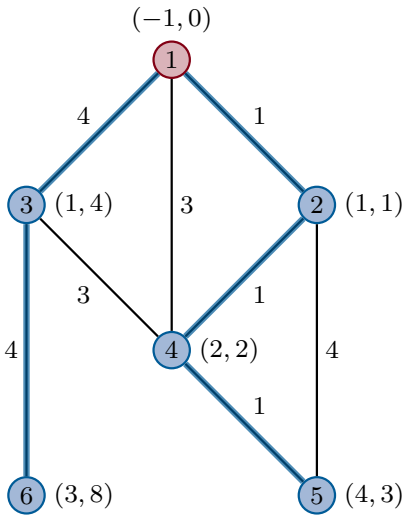
Algorithmus von Bellman-Ford

- $p[i]$: Vorgänger von Knoten i im Graphen

- $d[i]$: Distanz von der Wurzel zu Knoten i

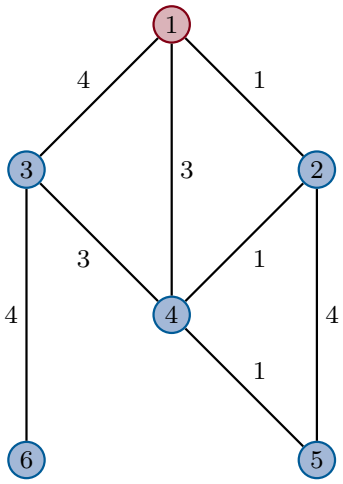
// Initialisierung
for $i \in \mathcal{N}$ **do**
  $p[i] = -1$
  $d[i] = \begin{cases} 0 & i = s \\ \infty & \text{sonst} \end{cases}$
end for
$T = \{s\}$ // Menge der erreichbaren Knoten

// Berechnung der Pfade
while $d[j]$ changes for some $j \in \mathcal{N}$ **do**
  $S = \{\}$ // Aktualisierte Knoten
  for $i \in T$ **do**
    // Für alle Nachbarn von i:
    for $\forall j : (i, j) \in \mathcal{E}$ **do**
      if $d[i] + c_{ij} < d[j]$ **then**
        $p[j] = i$
        $d[j] = d[i] + c_{ij}$
        $S = S \cup \{j\}$
      end if
    end for
  end for
  $T = T \cup S$
end while



Dijkstras Algorithmus

- $p[i]$: Vorgänger von Knoten i im Graphen
- $d[i]$: Distanz von der Wurzel zu Knoten i
- Priority Queue Q, welche Elemente nach Schlüsseln sortiert ausgibt

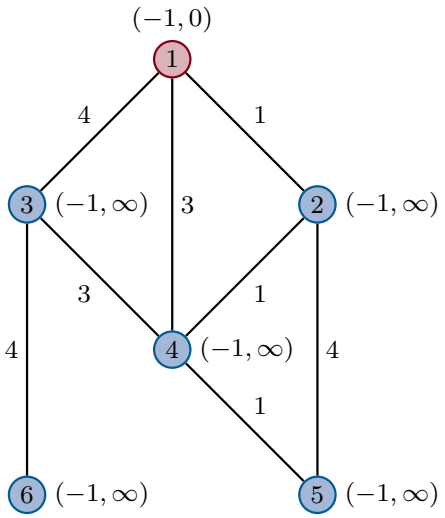


Dijkstras Algorithmus

- $p[i]$: Vorgänger von Knoten i im Graphen
- $d[i]$: Distanz von der Wurzel zu Knoten i
- Priority Queue Q, welche Elemente nach Schlüsseln sortiert ausgibt

for $i \in \mathcal{N}$ **do**
$p[i] = -1$
$d[i] = \begin{cases} 0 & i = s \\ \infty & \text{sonst} \end{cases}$

$Q.enqeue(i, d[i])$
end for
$T = \{\}$ // Menge der bearbeiteten Knoten

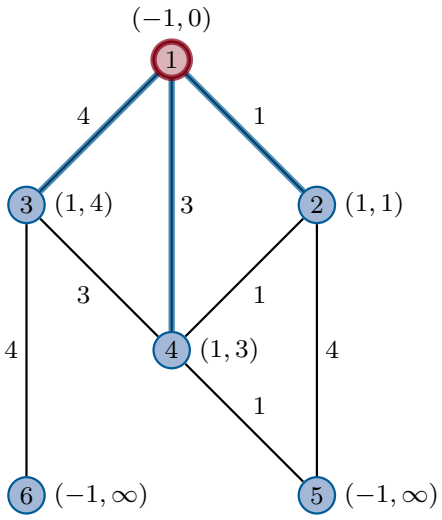


Dijkstras Algorithmus

- $p[i]$: Vorgänger von Knoten i im Graphen
- $d[i]$: Distanz von der Wurzel zu Knoten i
- Priority Queue Q, welche Elemente nach Schlüsseln sortiert ausgibt

for $i \in \mathcal{N}$ **do**
  $p[i] = -1$
  $d[i] = \begin{cases} 0 & i = s \\ \infty & \text{sonst} \end{cases}$
  $Q.enqeue(i, d[i])$
end for
$T = \{\}$ // Menge der bearbeiteten Knoten

// Berechnung der Pfade
while $T \neq \mathcal{N}$ **do**
  $i = Q.dequeue()$
  $T = T \cup \{i\}$
  // Für alle Nachbarn von i:
  for $\forall j : (i, j) \in \mathcal{E}$ **do**
    if $d[i] + c_{ij} < d[j]$ **then**
      $Q.decreaseKey(j, d[i] + c_{ij})$
      $p[j] = i$
    end if
  end for
end while

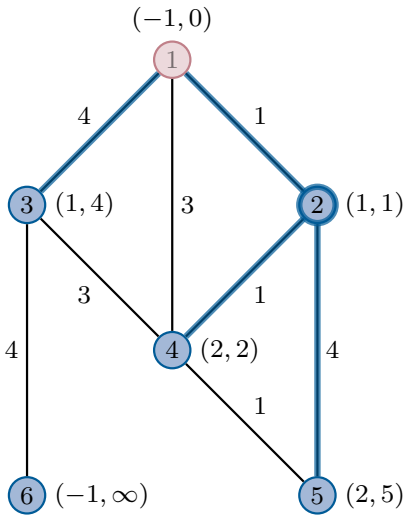


Dijkstras Algorithmus

- $p[i]$: Vorgänger von Knoten i im Graphen
- $d[i]$: Distanz von der Wurzel zu Knoten i
- Priority Queue Q, welche Elemente nach Schlüsseln sortiert ausgibt

for $i \in \mathcal{N}$ **do**
  $p[i] = -1$
  $d[i] = \begin{cases} 0 & i = s \\ \infty & \text{sonst} \end{cases}$
  $Q.enqeue(i, d[i])$
end for
$T = \{\}$ // Menge der bearbeiteten Knoten

// Berechnung der Pfade
while $T \neq \mathcal{N}$ **do**
  $i = Q.dequeue()$
  $T = T \cup \{i\}$
  // Für alle Nachbarn von i:
  for $\forall j : (i, j) \in \mathcal{E}$ **do**
    if $d[i] + c_{ij} < d[j]$ **then**
      $Q.decreaseKey(j, d[i] + c_{ij})$
      $p[j] = i$
    end if
  end for
end while

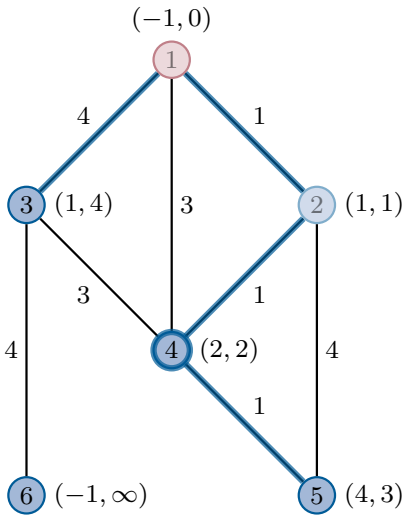


Dijkstras Algorithmus

- $p[i]$: Vorgänger von Knoten i im Graphen
- $d[i]$: Distanz von der Wurzel zu Knoten i
- Priority Queue Q, welche Elemente nach Schlüsseln sortiert ausgibt

for $i \in \mathcal{N}$ **do**
  $p[i] = -1$
  $d[i] = \begin{cases} 0 & i = s \\ \infty & \text{sonst} \end{cases}$
  $Q.enqeue(i, d[i])$
end for
$T = \{\}$ // Menge der bearbeiteten Knoten

// Berechnung der Pfade
while $T \neq \mathcal{N}$ **do**
  $i = Q.dequeue()$
  $T = T \cup \{i\}$
  // Für alle Nachbarn von i:
  for $\forall j : (i, j) \in \mathcal{E}$ **do**
    if $d[i] + c_{ij} < d[j]$ **then**
      $Q.decreaseKey(j, d[i] + c_{ij})$
      $p[j] = i$
    end if
  end for
end while

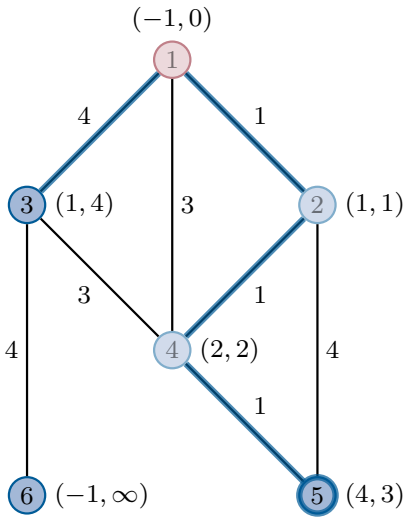


Dijkstras Algorithmus

- $p[i]$: Vorgänger von Knoten i im Graphen
- $d[i]$: Distanz von der Wurzel zu Knoten i
- Priority Queue Q, welche Elemente nach Schlüsseln sortiert ausgibt

for $i \in \mathcal{N}$ **do**
  $p[i] = -1$
  $d[i] = \begin{cases} 0 & i = s \\ \infty & \text{sonst} \end{cases}$
  $Q.enqeue(i, d[i])$
end for
$T = \{\}$ // Menge der bearbeiteten Knoten

// Berechnung der Pfade
while $T \neq \mathcal{N}$ **do**
  $i = Q.dequeue()$
  $T = T \cup \{i\}$
  // Für alle Nachbarn von i:
  for $\forall j : (i, j) \in \mathcal{E}$ **do**
    if $d[i] + c_{ij} < d[j]$ **then**
      $Q.decreaseKey(j, d[i] + c_{ij})$
      $p[j] = i$
    end if
  end for
end while

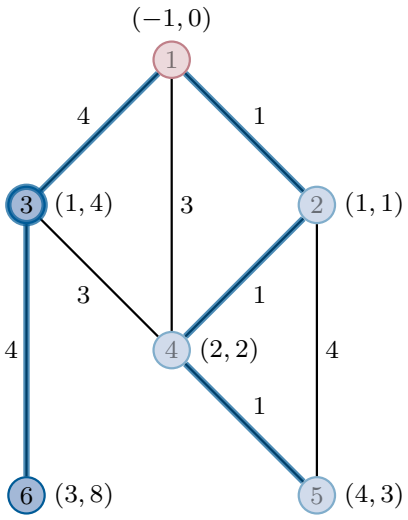


Dijkstras Algorithmus

- $p[i]$: Vorgänger von Knoten i im Graphen
- $d[i]$: Distanz von der Wurzel zu Knoten i
- Priority Queue Q, welche Elemente nach Schlüsseln sortiert ausgibt

for $i \in \mathcal{N}$ **do**
  $p[i] = -1$
  $d[i] = \begin{cases} 0 & i = s \\ \infty & \text{sonst} \end{cases}$
  $Q.enqeue(i, d[i])$
end for
$T = \{\}$ // Menge der bearbeiteten Knoten

// Berechnung der Pfade
while $T \neq \mathcal{N}$ **do**
  $i = Q.dequeue()$
  $T = T \cup \{i\}$
  // Für alle Nachbarn von i:
  for $\forall j : (i, j) \in \mathcal{E}$ **do**
    if $d[i] + c_{ij} < d[j]$ **then**
      $Q.decreaseKey(j, d[i] + c_{ij})$
      $p[j] = i$
    end if
  end for
end while

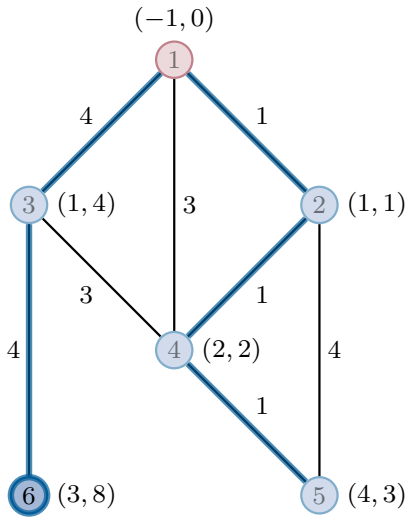


Dijkstras Algorithmus

- $p[i]$: Vorgänger von Knoten i im Graphen
- $d[i]$: Distanz von der Wurzel zu Knoten i
- Priority Queue Q, welche Elemente nach Schlüsseln sortiert ausgibt

for $i \in \mathcal{N}$ **do**
  $p[i] = -1$
  $d[i] = \begin{cases} 0 & i = s \\ \infty & \text{sonst} \end{cases}$
  $Q.enqeue(i, d[i])$
end for
$T = \{\}$ // Menge der bearbeiteten Knoten

// Berechnung der Pfade
while $T \neq \mathcal{N}$ **do**
  $i = Q.dequeue()$
  $T = T \cup \{i\}$
  // Für alle Nachbarn von i:
  for $\forall j : (i, j) \in \mathcal{E}$ **do**
    if $d[i] + c_{ij} < d[j]$ **then**
      $Q.decreaseKey(j, d[i] + c_{ij})$
      $p[j] = i$
    end if
  end for
end while



Eigenschaften des Algorithmus von Bellman-Ford:

- Im n-ten Durchlauf der while-Schleife werden alle Pfade der Länge höchstens n berücksichtigt
  (vergleiche min-plus-Produkt in n-ter Potenz)
- Keine komplexen Datenstrukturen notwendig
- Verteilte (dezentrale) Implementierung ohne Kenntnis der Topologie möglich (Beispiel)

Eigenschaften des Algorithmus von Dijkstra:

- Es werden immer Pfade über den im jeweiligen Schritt am günstigsten erreichbaren Knoten gesucht (Greedy-Prinzip)
- Wurde ein Knoten abgearbeitet, so ist garantiert, dass der kürzeste Pfad zu diesem Knoten gefunden ist
- Resourcenintensiver als der Algorithmus von Bellman-Ford, da komplexere Datenstrukturen notwendig (aber asymptotisch bessere Laufzeit)
- Vollständige Kenntnis der Netzwerktopologie erforderlich

Routing Information Protocol (RIP)

- Einfach Distanz-Vektor-Protokoll
- RIPv1 standardisiert in RFC 1058 (1988)
- Unterstützung für CIDR in RIPv2 hinzugefügt (RFC 2453, 1998)
- Einzige Metrik: Hop Count
- Hop Count Limit von 15, weiter entfernte Ziele sind nicht erreichbar

Funktionsweise:

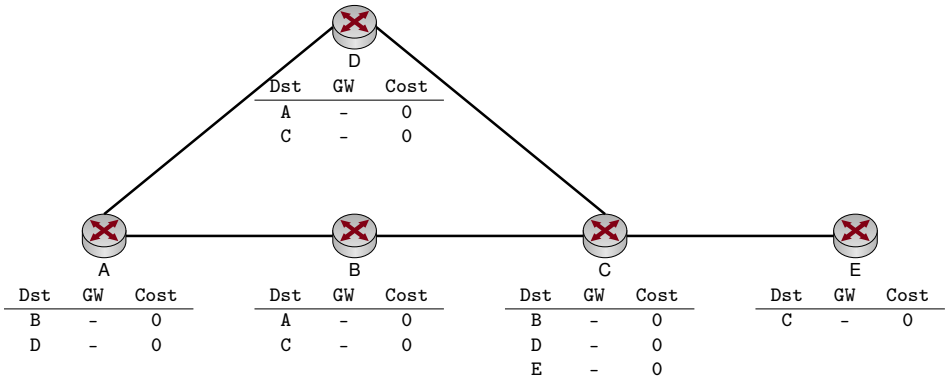
- Router senden in regelmäßigen Abständen (Standardwert 30 s) den Inhalt ihrer Routingtabelle an die Multicast-Adresse 224.0.0.9[6].

- Jeder RIP-Router akzeptiert diese Update-Nachrichten, inkrementiert die Metrik der enthaltenen Routen um 1 und vergleicht die Routen mit bereits vorhandenen Routen aus seiner Routingtabelle:
  - Enthält das Update eine noch unbekannte Route, wird diese in die eigene Routingtabelle übernommen
  - Enthält das Update eine Route zu einem bekannten Ziel aber mit niedrigeren Kosten, so wird die vorhandene Route durch das Update ersetzt
  - Andernfalls wird die vorhandene Route beibehalten

- Bleiben fünf aufeinanderfolgende Updates von einem Nachbarn aus, so werden alle Routen über diesen Next Hop aus der Routingtabelle entfernt.

Beispiel mit vereinfachter Darstellung:

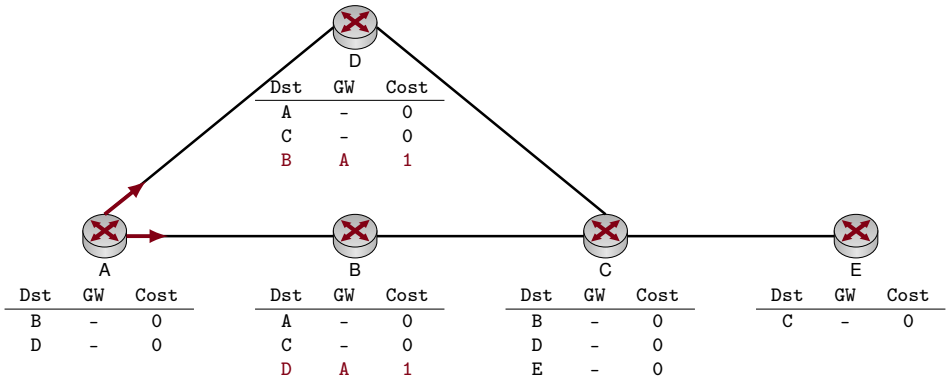
- Zwischen jeweils zwei Routern befindet sich sinnvollerweise ein Switch, welcher den Anschluss weiterer Computer an das jeweilige Subnetz ermöglicht
- Anstelle von IP- und Netzadressen tragen wir in die Routingtabellen lediglich die Namen der Router ein
- Für direkt erreichbare Nachbarn tragen wir kein Gateway ein
- Update-Nachrichten werden rundenweise verschickt (erst A, dann B, ...)



Beispiel mit vereinfachter Darstellung:

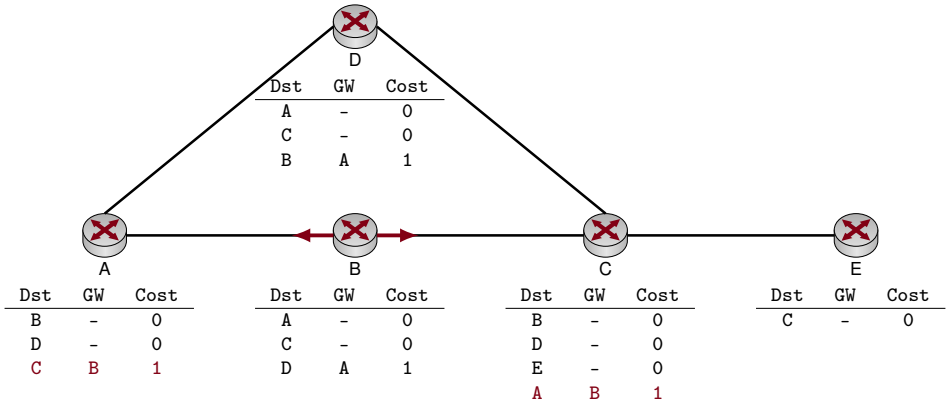
- Zwischen jeweils zwei Routern befindet sich sinnvollerweise ein Switch, welcher den Anschluss weiterer Computer an das jeweilige Subnetz ermöglicht
- Anstelle von IP- und Netzadressen tragen wir in die Routingtabellen lediglich die Namen der Router ein
- Für direkt erreichbare Nachbarn tragen wir kein Gateway ein
- Update-Nachrichten werden rundenweise verschickt (erst A, dann B, ...)

[6] Multicast wird im Rahmen der Vorlesung nicht näher behandelt. Stellen Sie sich die Funktionsweise hier wie einen Broadcast im lokalen Subnetz vor.



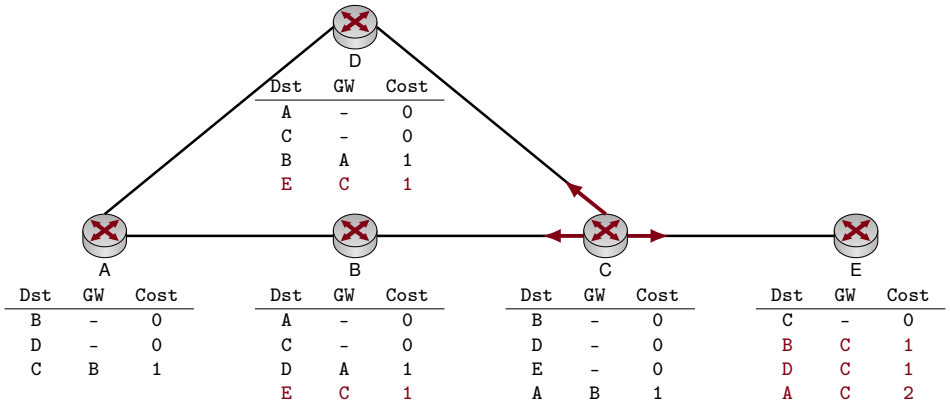
Beispiel mit vereinfachter Darstellung:

- Zwischen jeweils zwei Routern befindet sich sinnvollerweise ein Switch, welcher den Anschluss weiterer Computer an das jeweilige Subnetz ermöglicht
- Anstelle von IP- und Netzadressen tragen wir in die Routingtabellen lediglich die Namen der Router ein
- Für direkt erreichbare Nachbarn tragen wir kein Gateway ein
- Update-Nachrichten werden rundenweise verschickt (erst A, dann B, . . .)



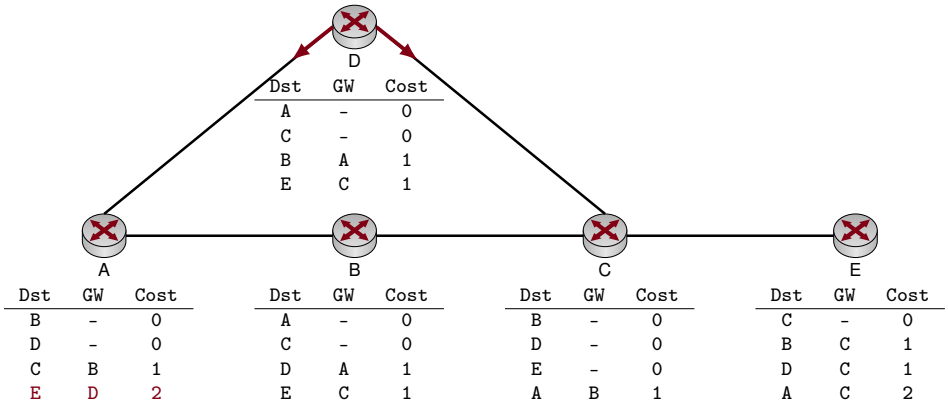
Beispiel mit vereinfachter Darstellung:

- Zwischen jeweils zwei Routern befindet sich sinnvollerweise ein Switch, welcher den Anschluss weiterer Computer an das jeweilige Subnetz ermöglicht
- Anstelle von IP- und Netzadressen tragen wir in die Routingtabellen lediglich die Namen der Router ein
- Für direkt erreichbare Nachbarn tragen wir kein Gateway ein
- Update-Nachrichten werden rundenweise verschickt (erst A, dann B, . . .)



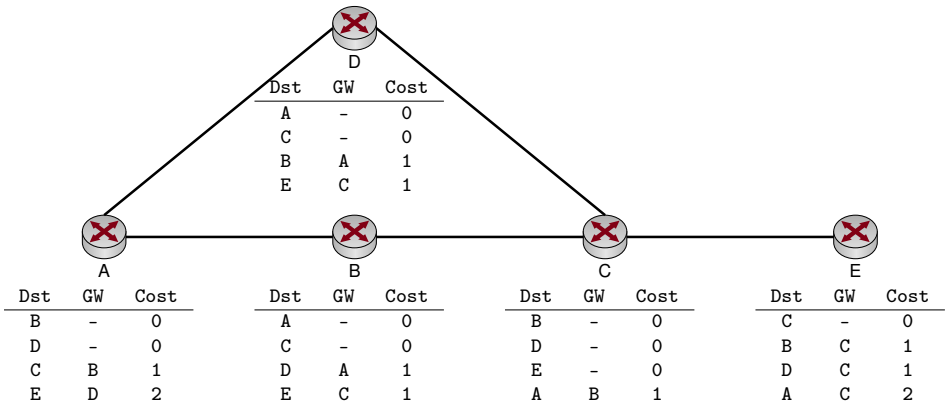
Beispiel mit vereinfachter Darstellung:

- Zwischen jeweils zwei Routern befindet sich sinnvollerweise ein Switch, welcher den Anschluss weiterer Computer an das jeweilige Subnetz ermöglicht
- Anstelle von IP- und Netzadressen tragen wir in die Routingtabellen lediglich die Namen der Router ein
- Für direkt erreichbare Nachbarn tragen wir kein Gateway ein
- Update-Nachrichten werden rundenweise verschickt (erst A, dann B, ...)



Beispiel mit vereinfachter Darstellung:

- Zwischen jeweils zwei Routern befindet sich sinnvollerweise ein Switch, welcher den Anschluss weiterer Computer an das jeweilige Subnetz ermöglicht
- Anstelle von IP- und Netzadressen tragen wir in die Routingtabellen lediglich die Namen der Router ein
- Für direkt erreichbare Nachbarn tragen wir kein Gateway ein
- Update-Nachrichten werden rundenweise verschickt (erst A, dann B, ...)



- Nach diesem Schritt kennt jeder Router eine kürzeste Route zu jedem anderen Router
- Da mehrere gleich lange Pfade existieren, bleibt es dem Zufall (der Reihenfolge der Updatenachrichten) überlassen, ob beispielsweise Router A als Next Hop zu E Router D oder B lernt

Das vorangegangene Beispiel lässt sich leicht auf gewichtete Kanten erweitern:

- Metrik ist nicht mehr die Anzahl der Hops zum Ziel, sondern das Kantegewicht

- Router addieren auf erhaltene Updates das Kantengewicht des Links, über den das Update empfangen wurde

⇒ Tutorübungen **Problem:**

- Bis jeder Router den besten Next Hop bestimmen kann, dauert es ggf. mehrere „Runden“
- Eine obere Schranke für die Anzahl der notwendigen Nachrichten, die jeder Router senden muss, ist die maximale Entfernung zwischen zwei Routern in Hops
- Die maximale Entfernung beträgt bei RIP 15 Hops
- Da Updates nur alle 30 s verschickt werden, ergibt sich eine maximale Verzögerung von $15 \cdot 30\,\text{s} = 7.5\,\text{min}$

Das vorangegangene Beispiel lässt sich leicht auf gewichtete Kanten erweitern:

- Metrik ist nicht mehr die Anzahl der Hops zum Ziel, sondern das Kantegewicht
- Router addieren auf erhaltene Updates das Kantengewicht des Links, über den das Update empfangen wurde

⇒ Tutorübungen **Problem:**

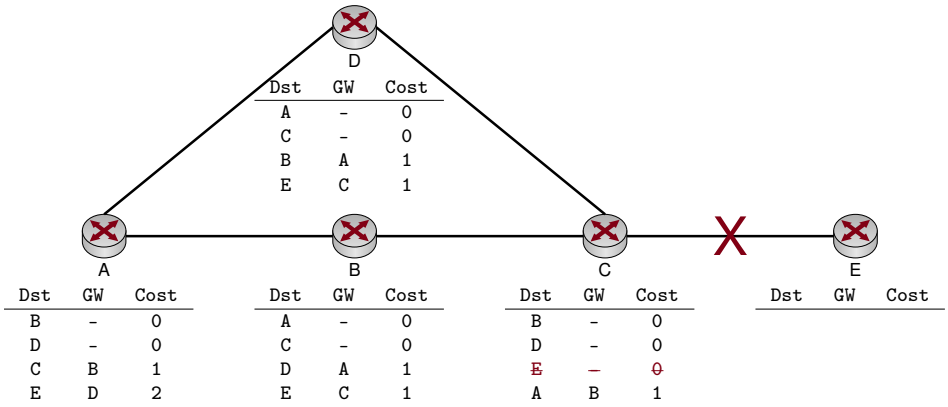
- Bis jeder Router den besten Next Hop bestimmen kann, dauert es ggf. mehrere „Runden“
- Eine obere Schranke für die Anzahl der notwendigen Nachrichten, die jeder Router senden muss, ist die maximale Entfernung zwischen zwei Routern in Hops
- Die maximale Entfernung beträgt bei RIP 15 Hops
- Da Updates nur alle 30 s verschickt werden, ergibt sich eine maximale Verzögerung von $15 \cdot 30\,\text{s} = 7.5\,\text{min}$

Lösung: Triggered Updates

- Sobald ein Router eine Änderung an seiner Routingtabelle vornimmt, sendet er sofort ein Update
- Dies führt zu einer Welle von Updates durch das Netzwerk
- Konvergenzzeit wird reduziert, aber das Netzwerk während der Updates ggf. stark belastet

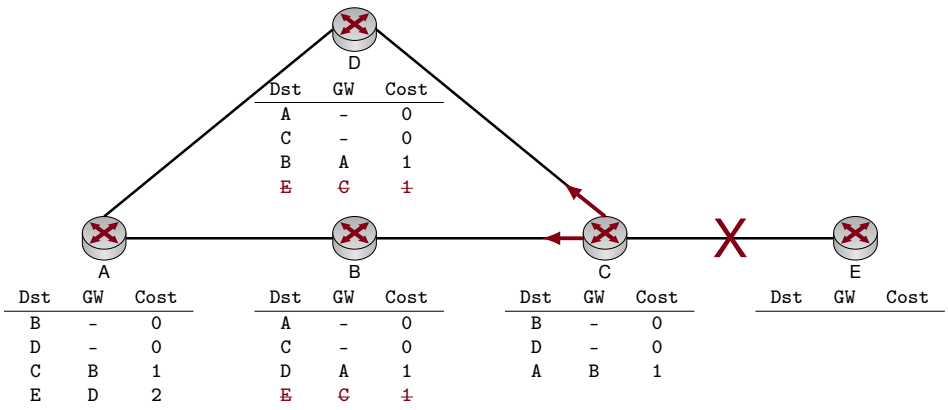
Anderes Problem: Count to infinity

- Link zwischen C und E fällt aus
- Reihenfolge, in der Updates versendet werden, ist dem Zufall überlassen



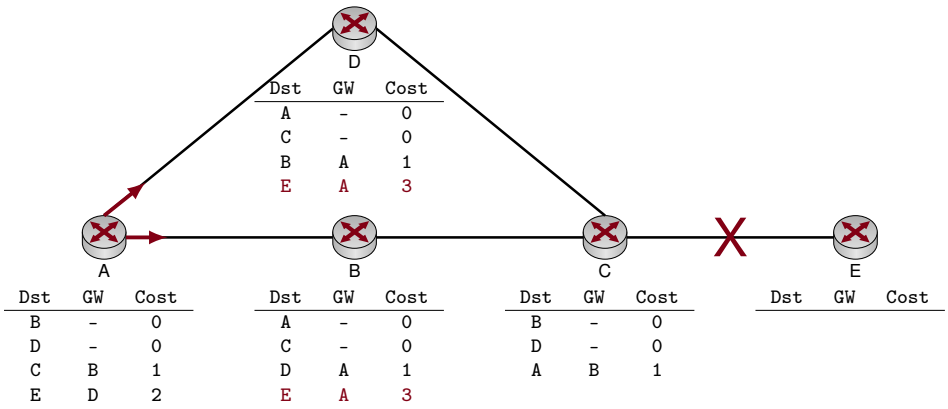
Anderes Problem: Count to infinity

- Link zwischen C und E fällt aus
- Reihenfolge, in der Updates versendet werden, ist dem Zufall überlassen



Anderes Problem: Count to infinity

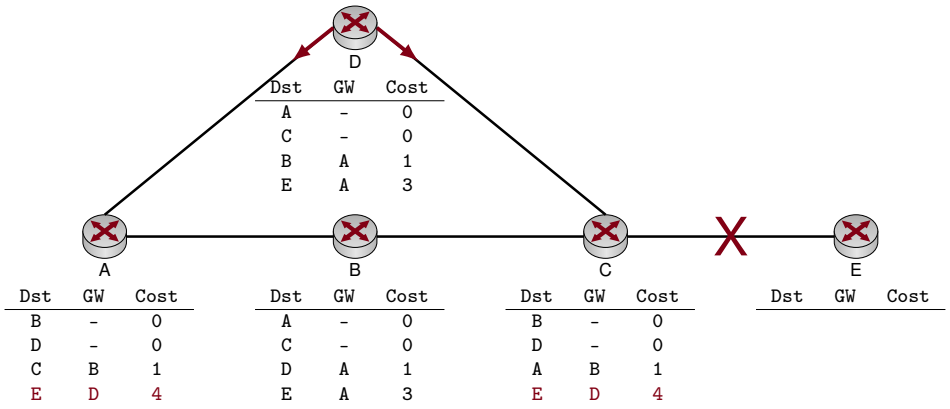
- Link zwischen C und E fällt aus
- Reihenfolge, in der Updates versendet werden, ist dem Zufall überlassen



Anderes Problem: Count to infinity

- Link zwischen C und E fällt aus

- Reihenfolge, in der Updates versendet werden, ist dem Zufall überlassen



Je nach Reihenfolge der Updates

- wird die fehlerhafte Route zu E weiterverbreitet und
- die Metrik stets incrementiert
- bis schließlich das Hop Count Limit von 15 erreicht ist.

Diesen Vorgang bezeichnet man als Count to infinity.

Lösungen:

- Split Horizon
  - Neue Regel: „Sende dem Nachbarn, von dem Du die Route zu X gelernt hast, keine Route zu X“
  - Im vorherigen Beispiel würde A die Route zu E nicht an D, wohl aber an B schicken
  - Split Horizon verbessert die Situation, kann das Problem aber nicht lösen
- Poison Reverse
  - Anstelle dem Nachbarn, von dem eine Route zu X gelernt wurde, keine Route zu X mehr zu schicken, wird eine Route mit unendlicher Metrik gesendet
  - Im vorherigen Beispiel würde A die Route zu E an D mit Metrik 15 schicken
  - Die fehlerhafte Route würde nach wie vor B erreichen
  - Auch Poison Reverse kann das Problem nicht vollständig lösen
- Path Vector
  - Sende bei Updates nicht nur Ziel und Kosten, sondern auch den vollständigen Pfad, über den das Ziel erreicht wird
  - Jeder Router prüft vor Installation der Route, ob er selbst in diesem Pfad bereits vorhanden ist
  - Falls ja, handelt es sich um eine Schleife und das Update wird verworfen
  - Path Vector verhindert Routing Loops und damit auch Count to Infinity, vergrößert jedoch die Update-Nachrichten und die Protkollkomplexität

Übersicht: Ausgewählte Routing-Protokolle Distanz-Vektor-Protokolle

- RIP (Routing Information Protocol)
  Sehr einfaches Protokoll, Hop-Count als einzige Metrik, geeignet für eine geringe Anzahl von Netzen, wird von den meisten Routern unterstützt (sogar einige Heimgeräte).
- IGRP (Interior Gateway Routing Protocol)
  Proprietäres Routing Protokoll von Cisco, unterstützt komplexere Metriken als RIP

- EIGRP (Enhanced Interior Gateway Routing Protocol)
  Proprietäres Routing Protokoll von Cisco, Nachfolger von IGRP, deutlich verbesserte Konvergenzeigenschaften

- AODV (Ad hoc On-Demand Distance Vector)
  Einsatz in kabellosen vermaschten Netzwerken, Routen werden nicht proaktiv ausgetauscht sondern on-demand gesucht (reaktives Protokoll)

Link-State-Protokolle

- OSPF (Open Shortest Path First)
  Industriestandard für mittlere bis große Anzahl von Netzwerken

- IS-IS (Intermediate System to Intermediate System)
  Seltener eingesetztes, leistungsfähiges Routingprotokoll, welches unabhängig von IP ist (es handelt sich um ein ISO-standardisiertes Layer-3-Protokoll)

- HWMP (Hybrid Wireless Mesh Protocol)
  Ermöglicht Routing in IEEE 802.11s (Wireless Mesh Networks)

3.4.3. Autonome Systeme

Autonome Systeme Alle bislang vorgestellten Routingprotokolle

- bestimmen beste Pfade anhand objetiver Kriterien (Hopcount, Bandbreite, Delay, ...),
- bieten aber keine bzw. nur eingeschränkte Möglichkeiten, Routen direkt zu beeinflussen.

Manchmal ist es aber wünschenswert, Routen auf Basis anderer Kriterien zu wählen:

- Tatsächlich anfallende monetäre Kosten
- Netze / Länder, durch die Datenverkehr zu einem Ziel weitergeleitet wird
- Infrastrukturentscheidungen (z. B. Belastung einzelner Router)

Routingentscheidungen auf Basis derartiger Kriterien bezeichnet man als Policy-Based Routing.

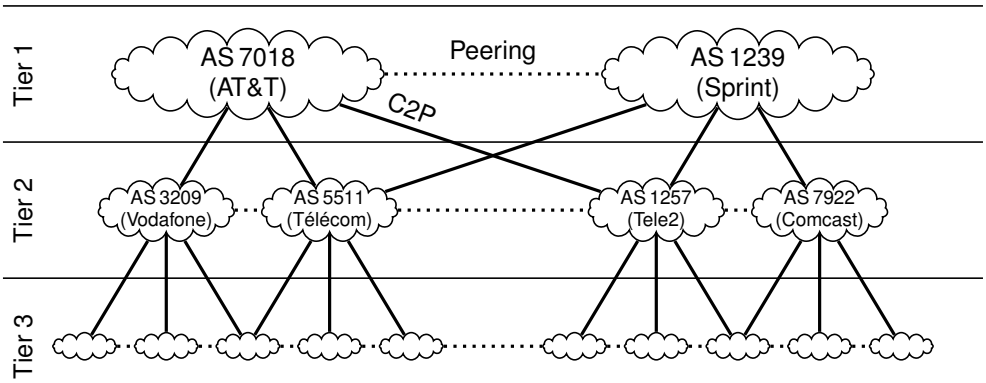
Definition: Autonomes System:

Eine Menge von Netzwerken, die unter einheitlicher administrativer Kontrolle stehen, bezeichnet man als Autonomes System (AS). Ein AS wird i. d. R. durch einen 16 Bit Identifier, der sog. AS-Nummer identifiziert. Beim Einsatz von Routingprotokollen wird unterschieden:

- Innerhalb eines autonomen Systems werden Interior Gateway Protocols (IGPs) wie RIP, OSPF, EIGRP oder IS-IS eingesetzt.
- Zum Austausch von Routen zwischen Autonomen Systemen wird ein Exterior Gateway Protocol (EGP) verwendet.

Das einzige in der Praxis verwendete EGP ist das Border Gateway Protocol (BGP), und zwar in der Version BGP4.

Internet: Stark vereinfachte schematische Darstellung [[Image Discarded Due To '/tikz/external/mode=list and make']]



- Autonome Systeme können durch Upstream-Provider oder durch Peering miteinander verbunden sein.
- An Internet Exchange Points, z. B. DE-CIX, existieren zahlreiche Peering-Verbindungen.
- Peering-Verbindungen sind aus Kostengründen gegenüber Customer-Provider (C2P) Verbindungen zu bevorzugen.
- Die Border-Router eines AS „announcen“ Präfixe, die über dieses AS erreichbar sind.
- AS 5511 announced seine eigenen Customer an seine Peerings und Upstream-Provider.
- AS 5511 würde evtl. die Netze von Vodafone an Tele2 announcen, sicher aber nicht an Sprint oder AT&T.

Faustregel: Für vertikale Verbindungen muss der jeweilige Customer („kleinere Provider“) bezahlen, weshalb Horizontaler Verbindungen (Peerings) bevorzugt werden.

3.5. Nachfolge von IP(v4): IPv6

3.5.0.1. Nachfolge von IP(v4): IPv6

Wir haben bereits gesehen, dass

- der IPv4 Adressraum aus heutiger Sicht zu knapp bemessen ist und
- historisch bedingt Fehler bei der Adressvergabe gemacht wurden.

Außerdem ist die Verarbeitung des IPv4 Headers unnötig komplex:

- Header variabler Länge ($\Rightarrow$ Angabe der Header- und Datenlänge)
- Ungenutzte bzw. im Laufe der Zeit redefinierte Felder (TOS, DSCP, ECN)
- Fragmentierung und Reassemblierung bedeutet erheblichen Mehraufwand

IPv6 begegnet diesen Problemen:

- 128 Bit Adressraum $\Rightarrow 2^{128} \approx 10^{38}$ Adressen
  Das sind etwa $6.67 \cdot 10^{23}$ Adressen pro m^2 Erdoberfläche (mit Wasserflächen)!
- Header fester Länge für schnelle Verarbeitung
- Erweiterbarkeit durch sog. Extension Headers
- Keine Fragmentierung von IP-Paketen mehr (Anpassung der MTU beim Sender)
- Viele neue Features zur automatischen Konfiguration, Adressvergabe, Auffindung lokaler Gateways, etc.

Notation IPv6 Adressen werden infolge ihrer Länge kompakt

- in hexadezimaler Schreibweise
- in Gruppen zu je 16 Bit
- getrennt durch : dargestellt.

Beispiel:

`2001:4ca0:2001:0011:2a37:37ff:fe02:3241 / 64`

- Wie IPv4-Adressen bestehen IPv4-Adressen aus einem Netz- und Hostanteil variabler Länge
- Die von IPv4-Adressen bekannte Präfixschreibweise wurde übernommen

Abkürzende Schreibweisen:

- In jedem Block von 16 Bit Länge können führende Nullen weggelassen werden
- Bestehen einer oder mehrere aufeinanderfolgende Blöcke nur aus Nullen, so können diese durch :: abgekürzt werden:

`fe80::2a37:37ff:fe02:3241 / 64 = fe80:0000:0000:0000:2a37:37ff:fe02:3241 / 64`

IPv6-Header (ohne Extension Headers) Version

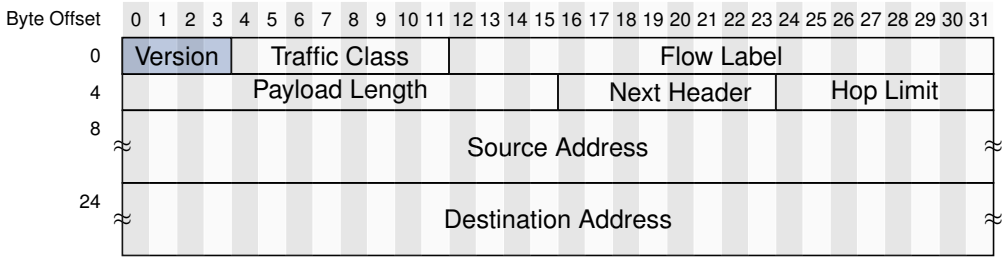


Abbildung 3.16.: IPv6-Header (minimale Länge: 40 Byte)

- Gibt die verwendete IP-Version an.
- Gültige Werte sind 6 (IPv6) und 4 (IPv4).

IPv6-Header (ohne Extension Headers) Traffic Class

- Äquivalent zum TOS-Feld des IPv4 Headers.
- Wird zur Verkehrspriorisierung (QoS) eingesetzt.

IPv6-Header (ohne Extension Headers) Flow Label

- Ebenfalls zur Verkehrspriorisierung eingesetzt.

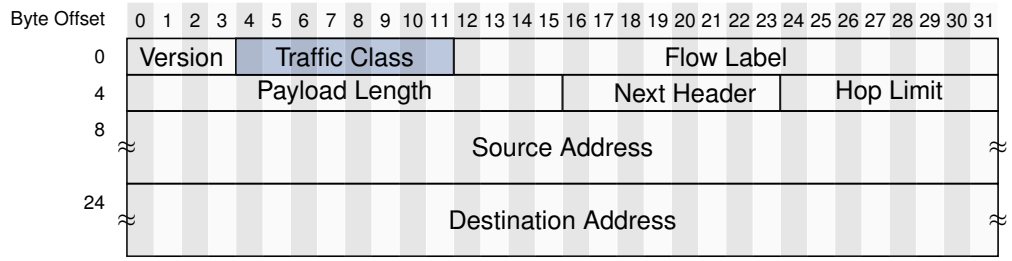


Abbildung 3.17.: IPv6-Header (minimale Länge: 40 Byte)

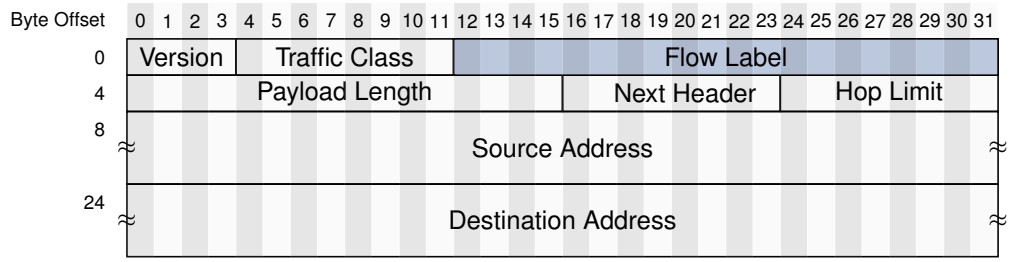


Abbildung 3.18.: IPv6-Header (minimale Länge: 40 Byte)

- Ermöglicht es, zu demselben Datenstrom (Flow) gehörende Pakete auf Schicht 3 zu identifizieren und gleich zu behandeln.

IPv6-Header (ohne Extension Headers) Payload Length

- Länge der Daten inkl. möglicherweise vorhandener Extension Headers.
- Angabe in Vielfachen von 1 Byte.

IPv6-Header (ohne Extension Headers) Next Header

- Identifiziert den Typ des nächsten Headers.
- Dieser kann entweder ein Extension Header von IPv6 sein oder
- der Header der im Paket transportierten Daten (z. B. ICMP-, TCP-, UDP-Header).

IPv6-Header (ohne Extension Headers) Hop Limit

- Entspricht dem TTL-Feld des IPv4-Headers.
- Wird beim Weiterleiten des Pakets durch einen Router um 1 dekrementiert.
- Erreicht das Feld den Wert 0, wird das Paket verworfen und ein ICMPv6 Time Exceeded an den Absender zurückgeschickt.

IPv6-Header (ohne Extension Headers) Source Address

- 128 Bit (16 Byte) lange Absenderadresse.

IPv6-Header (ohne Extension Headers) Destination Address

- 128 Bit (16 Byte) lange Zieladresse.

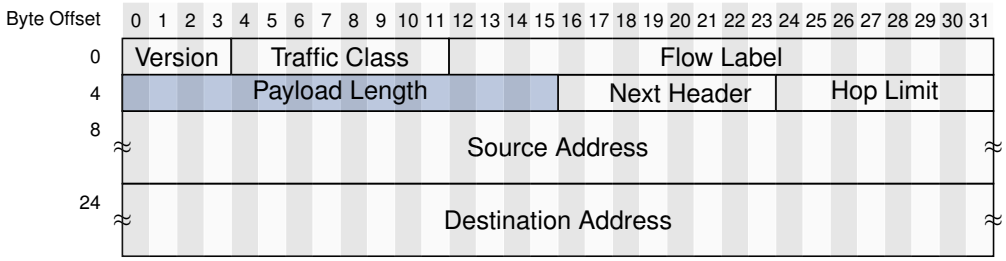


Abbildung 3.19.: IPv6-Header (minimale Länge: 40 Byte)

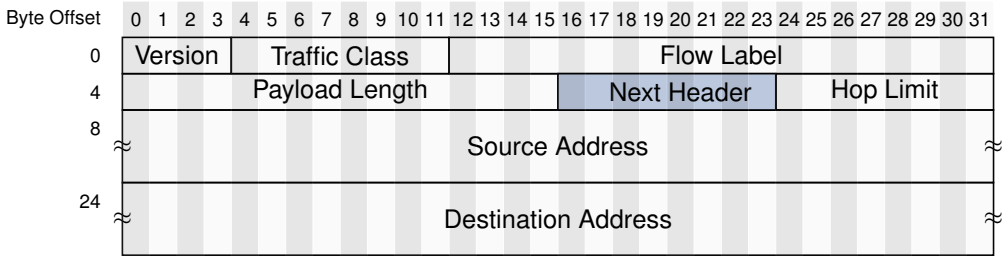


Abbildung 3.20.: IPv6-Header (minimale Länge: 40 Byte)

Kompatibilität:

- IPv4 und IPv6 sind nicht kompatibel, können aber nebeneinander existieren
- Knoten verwenden heute häufig eine IPv4 und eine IPv6 Adresse
- Router müssen für beide Protokollversionen Routinginformationen getrennt voneinander austauschen und verarbeiten

Stand heute:

- Obwohl IPv6 bereits seit 1998 standardisiert ist (RFC 2460), ist die Umstellung auf IPv6 noch lange nicht abgeschlossen.
- Der mit Abstand größte Teil des weltweiten Datenverkehrs ist noch immer IPv4:

3.5.0.2. Zusammenfassung

In diesem Kapitel haben wir

- die Vorteile von Paketvermittlung gegenüber Leitungs- und Nachrichtenvermittlung erarbeitet,
- die Notwendigkeit logischer Adressen zur End-zu-End Adressierung erkannt,
- zwei unterschiedliche Protokolle zur End-zu-End Adressierung im Internet kennen gelernt,
- Methoden zur weiteren logischen Unterteilung von Netzen in Subnetze kennengelernt und
- ein grundlegendes Verständnis bzgl. des Austauschs von Routinginformationen im Internet entwickelt.

Was wir wissen sollten:

- Was sind die Unterschiede zwischen Leitungs-, Nachrichten- und Paketvermittlung?
- Worin besteht der technische und logische Unterschied zwischen MAC- und IP-Adressen?
- Wie werden IP-Adressen in Netz- und Hostanteil aufgeteilt?

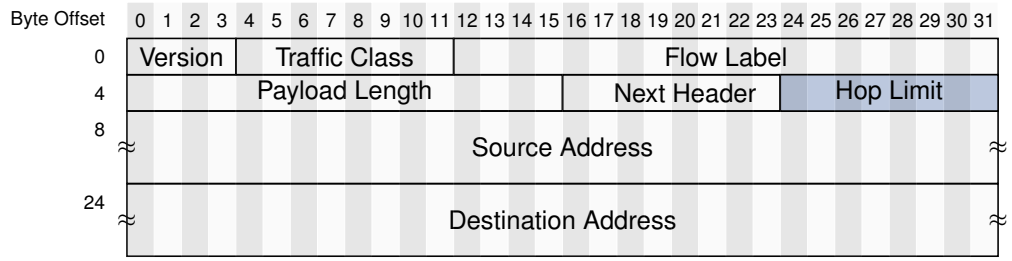


Abbildung 3.21.: IPv6-Header (minimale Länge: 40 Byte)

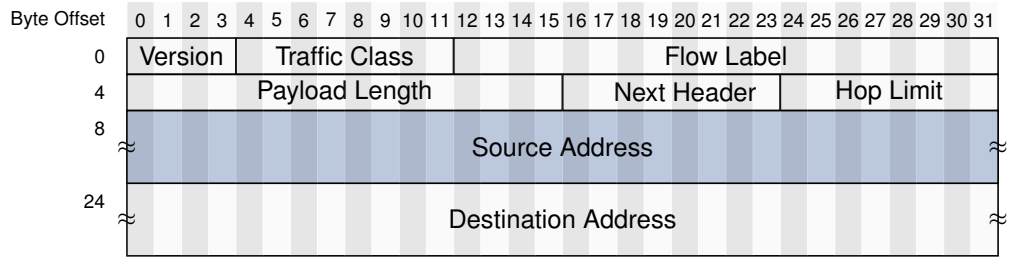


Abbildung 3.22.: IPv6-Header (minimale Länge: 40 Byte)

- Wie werden IP-Adressen in MAC-Adressen übersetzt?
- Was ist eine Routing Tabelle?
- Wie treffen Hosts und Router Weiterleitungsentscheidungen (Forwarding)?
- Wie tauschen Router untereinander Routinginformationen aus?
- Welche grundlegenden Typen von Routingprotokollen gibt es?
- Was sind die wesentlichen Unterschiede zwischen IPv4 und IPv6?

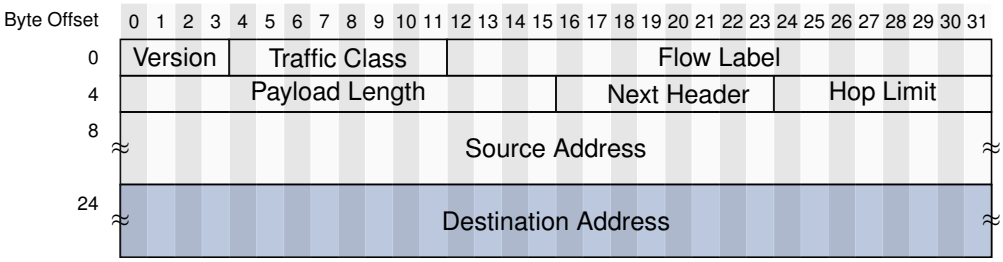
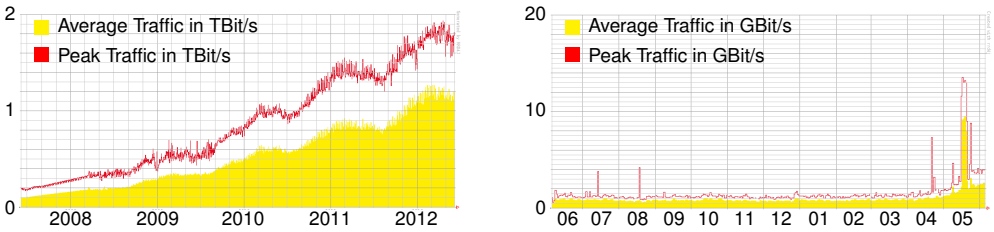


Abbildung 3.23.: IPv6-Header (minimale Länge: 40 Byte)



(a) IPv4-Verkehr 2008 - 2012

(b) IPv6-Verkehr 2011 - 2012

Abbildung 3.24.: IPv4- und IPv6-Datenverkehr am DE-CIX [2]

4. Transportschicht

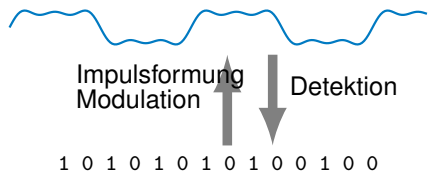
4.1. Motivation

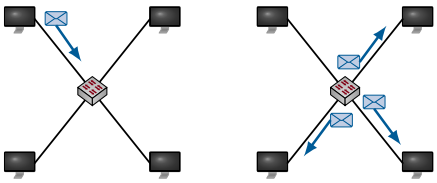
4.1.0.3. Wir haben bislang gesehen:

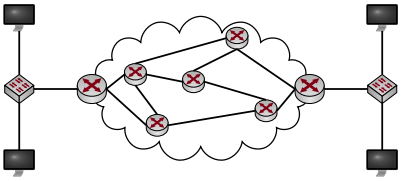
- Wie digitale Daten durch messbare Größen dargestellt, übertragen und rekonstruiert werden (Schicht 1)

- Wie der Zugriff auf das Übertragungsmedium gesteuert und der jeweilige Next-Hop adressiert wird (Schicht 2)

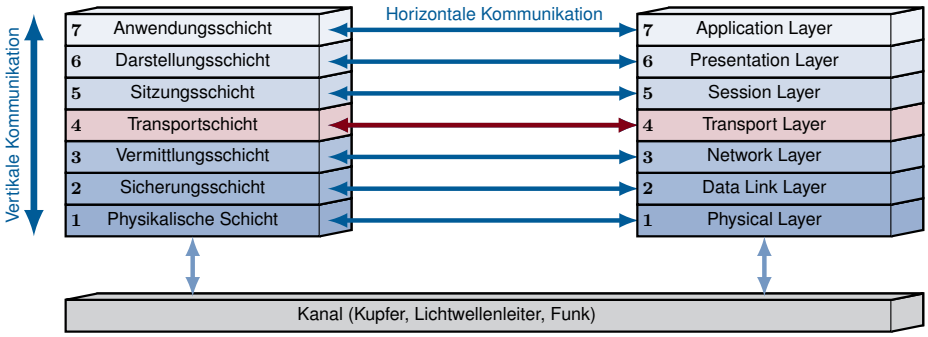
- Wie auf Basis logischer Adressen End-zu-End Verbindungen zwischen Sender und Empfänger hergestellt werden







4.1.0.4. Einordnung im ISO/OSI-Modell



4.1.0.5. Aufgaben der Transportschicht

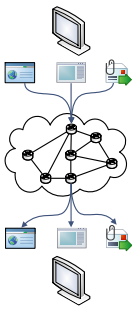
:

Die wesentlichen Aufgaben der Transportschicht sind

- Multiplexing von Datenströmen unterschiedlicher Anwendungen bzw. Anwendungsinstanzen,

Multiplexing:

- Segmentierung der Datenströme unterschiedlicher Anwendungen (Browser, Chat, Email, ...)
- Segmente werden in jeweils unabhängigen IP-Paketen zum Empfänger geroutet
- Empfänger muss die Segmente den einzelnen Datenströmen zuordnen und an die jeweilige Anwendung weiterreichen



:

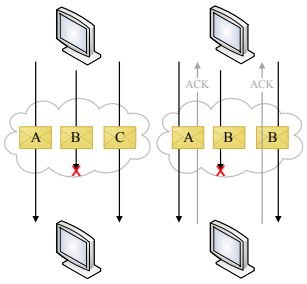
Die wesentlichen Aufgaben der Transportschicht sind

- Multiplexing von Datenströmen unterschiedlicher Anwendungen bzw. Anwendungsinstanzen,

- Bereitstellung verbindungsloser und verbindungsorientierter Transportmechanismen und

Transportdienste:

- Verbindungslos (Best Effort)
  - Segmente sind aus Sicht der Transportschicht voneinander unabhängig
  - Keine Sequenznummern, keine Übertragungswiederholung, keine Garantie der richtigen Reihenfolge
- Verbindungsorientiert
  - Übertragungswiederholung bei Fehlern
  - Garantie der richtigen Reihenfolge einzelner Segmente



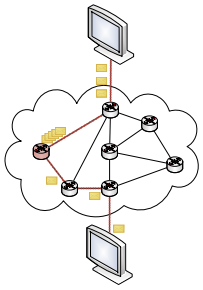
:

Die wesentlichen Aufgaben der Transportschicht sind

- Multiplexing von Datenströmen unterschiedlicher Anwendungen bzw. Anwendungsinstanzen,
- Bereitstellung verbindungsloser und verbindungsorientierter Transportmechanismen und
- Mechanismen zur Stau- und Flusskontrolle.

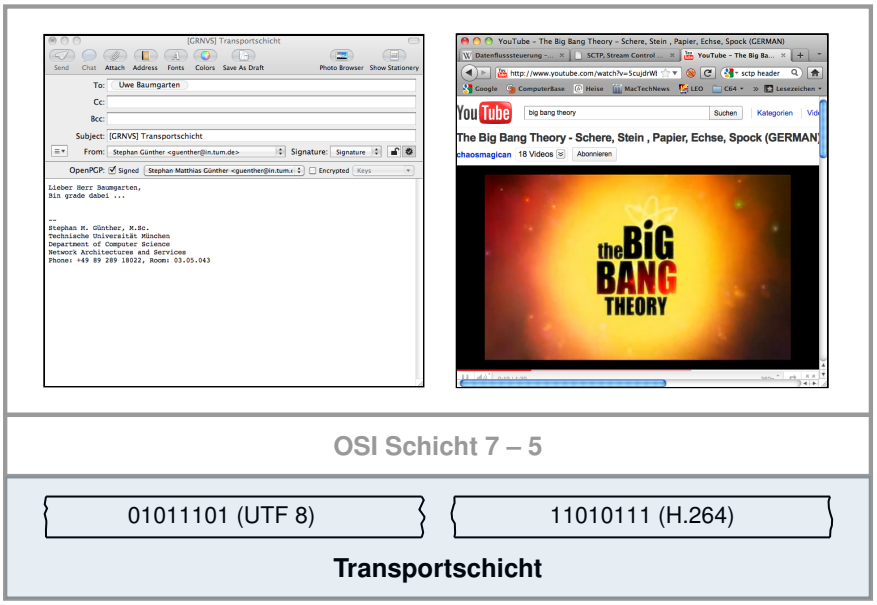
Stau- und Flusskontrolle:

- Staukontrolle (Congestion Control)
  - Reaktion auf drohende Überlast im Netz
- Flusskontrolle (Flow Control)
  - Laststeuerung durch den Empfänger



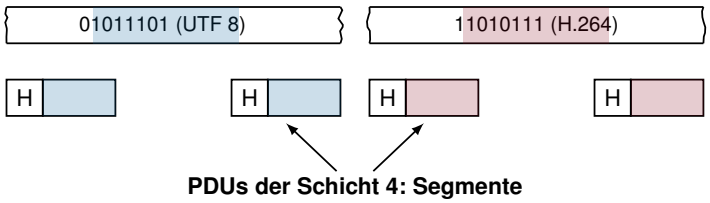
4.2. Multiplexing

4.2.0.6. Multiplexing



Auf der Transportschicht

1. werden die kodierten Datenstrome in Segmente unterteilt und
2. jedes Segment mit einem Header versehen.



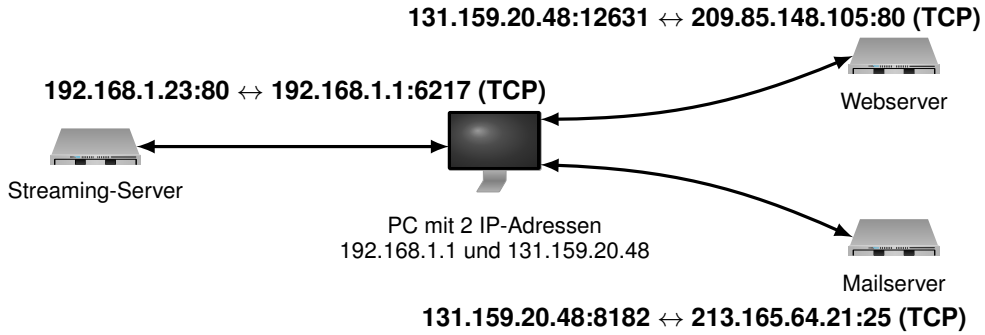
Ein solcher Header enthält jeweils mindestens

- einen Quellport und
- einen Zielport,

welche zusammen mit den IP-Adressen und dem verwendeten Transportprotokoll die Anwendung auf dem jeweiligen Host eindeutig identifizieren.

$$\Rightarrow \textbf{5-Tupel (SrcIPAddr, SrcPort, DstIPAddr, DstPort, Protocol)}$$

Beispiel:



- Portnummern sind bei den bekannten Transportprotokollen 16 Bit lang
- Betriebssysteme verwenden das 5-Tupel (IP-Adressen, Portnummern, Protokoll), um Anwendungen Sockets bereitzustellen
- Eine Anwendung wiederum adressiert einen Socket mittels eines File-Deskriptors (ganzzahliger Wert)
- Verbindungsorientierte Sockets können nach dem Verbindungsaufbau sehr einfach genutzt werden, da der Empfänger bereits feststeht (Lesen und Schreiben mittels Systemaufrufen `read()` und `write()` möglich)
- Verbindungslose Sockets benötigen Adressangaben, an wen gesendet oder von wem empfangen werden soll (`sendto()` und `recvfrom()`)

4.3. Verbindungslose Übertragung

4.3.0.7. Verbindungslose Übertragung

Funktionsweise: Header eines Transportprotokolls besteht mind. aus

- Quell- und Zielport sowie
- einer Längenangabe der Nutzdaten.

Dies ermöglicht es einer Anwendung beim Senden für jedes einzelne Paket

- den Empfänger (IP-Adresse) und
- die empfangende Anwendung (Protokoll und Zielport) anzugeben.

Probleme: Da die Segmente unabhängig voneinander und aus Sicht der Transportschicht zustandslos versendet werden, kann nicht sichergestellt werden, dass

- Segmente den Empfänger erreichen (Pakete können verloren gehen) und
- der Empfänger die Segmente in der richtigen Reihenfolge erhält (Pakete werden unabhängig geroutet).

:

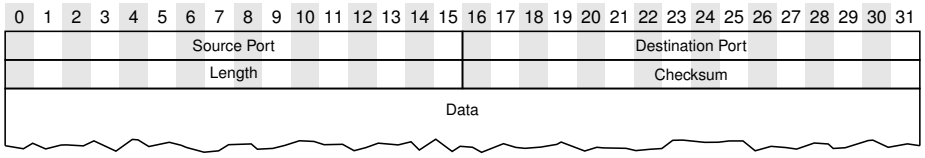
Folglich spricht man von einer ungesicherten, verbindungslosen oder nachrichtenorienterten Kommunikation. (Nicht zu verwechseln mit nachrichtenorientierter Übertragung auf Schicht 2)

Verbindungslose POSIX-Sockets werden mittels des Präprozessormakros `SOCK_DGRAM` identifiziert.

Case Study: User Datagram Protocol (UDP) Das User Datagram Protcol (UDP) ist eines von zwei gebräuchlichen Transportprotokollen im Internet. Es bietet

- ungesicherte / nachrichtenorientierte Übertragung
- bei geringem Overhead.

UDP-Header:



- „Length“ gibt die Länge von Header und Daten in Vielfachen von Byte an
- Die Checksumme erstreckt sich über Header und Daten
  - Die UDP-Checksumme ist optional
  - Wird sie nicht verwendet, wir das Feld auf 0 gesetzt
  - Wird sie verwendet, wird zur Berechnung ein Pseudo-Header genutzt (eine Art „Default-IP-Header“ der nur zur Berechnung der Prüfsumme dient). Er beinhaltet folgende Felder des IP-Headers: Quell- und Ziel-IP-Adresse, ein 8-Bit-Feld mit Nullen, Protocol-ID und Länge des UDP-Datagramms.

Vorteile von UDP:

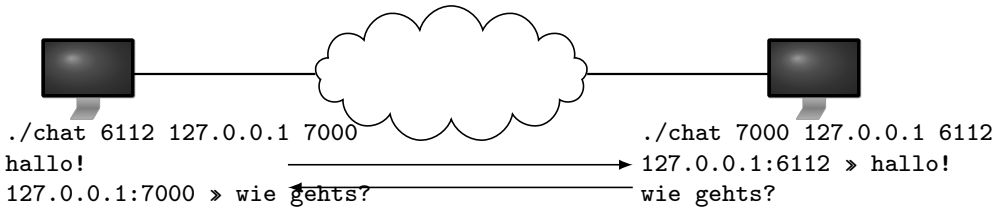
- Geringer Overhead
- Keine Verzögerung durch Verbindungsaufbau oder Retransmits und Reordering von Segmenten
- Gut geeignet für Echtzeitanwendungen (Voice over IP, Online-Spiele) sofern gelegentlicher Paketverlust in Kauf genommen werden kann
- Keine Beeinflussung der Datenrate durch Fluss- und Staukontrollmechanismen (kann Vorteile haben, siehe Übung)

Nachteile von UDP:

- Es wird keinerlei Dienstqualität zugesichert (beliebig hohe Fehlerrate)
- Kein Reordering von Segmenten
- Keine Flusskontrolle (schneller Sender kann langsamen Empfänger überfordern)
- Keine Staukontrollmechanismen (Überlast im Netz führt zu hohen Verlustraten)

Case Study: UDP-Chat Was wir wollen:

- Eine Anwendung, die gleichzeitig als Client und Server arbeiten soll (P2P-Modell)
- Nur 1:1-Verbindungen, also keine Gruppen-Chats



Was wir brauchen: Einen Socket,

- auf dem ausgehende Nachrichten gesendet werden (an Ziel-IP und Ziel-Port) und
- der an die lokale(n) IP(s) und Portnummer gebunden wird, um Nachrichten empfangen zu können

Welche Sprache?

- C natürlich (;

Wichtige structs

```
struct sockaddr_in {
__kernel_sa_family_t sin_family; /* Address family */
__be16               sin_port;   /* Port number */
struct in_addr       sin_addr;   /* Internet address */
/* Pad to size of ‘struct sockaddr’. */
unsigned char     __pad[__SOCK_SIZE__ - sizeof(short int) -
sizeof(unsigned short int) - sizeof(struct in_addr)];
};
struct in_addr {
__be32  s_addr;
};
```

Code für unser Programm:

```
struct sockaddr_in local;
local.sin_family       = AF_INET;
local.sin_port         = htons(LOCALPORT);  // anwendungsspezifischer Port
local.sin_addr.s_addr  = INADDR_ANY;        // empfange von allen Adressen
```

Sockets

- Aus Sicht des Betriebssystems ist ein Socket nichts weiter als ein File-Deskriptor.
- Sockets stellen die Schnittstelle zwischen einem Programm (unserer Chatanwendung) und dem Betriebssystem dar.

Ein Socket für unser Programm:

```
int sd;
if (0 > (sd=socket(AF_INET,SOCK_DGRAM,IPPROTO_UDP))) {
perror("socket() failed");
exit(1);
}
```

Der Socket muss noch eine Adresse bekommen:

```
if (0 > bind(sd,(struct sockaddr *)&local,sizeof(local))) {
perror("bind() failed");
exit(1);
}
```

Wie merkt unser Programm, wenn neue Daten ankommen? Hier gibt es 3 Möglichkeiten:

- Einfach ein `read()` auf den Socket
  - `read()` blockiert, solange bis etwas kommt
  - Mit einem einzelnen Prozess bzw. Thread können wir so nur einen einzigen Socket überwachen
  - Unser Programm würde nicht einmal auf Tastatureingaben reagieren
- Der scheinbar komplizierte Weg über `select()`
  - Wir packen alle File-Deskriptoren, die überwacht werden sollen, in ein Set
  - Wir übergeben `select()` dieses Set
  - Sobald etwas passiert, gibt uns `select()` ein Set mit genau den File-Deskriptoren zurück, die bereit sind
- Event-Loop-basiert, z.B. mit "epoll" (wird hier nicht behandelt)

Ein select() für unser Programm:

```
fd_set rfds,rfd;
FD_ZERO(&rfds);
FD_SET(STDIN_FILENO,&rfds);
FD_SET(sd,&rfds);
maxfd = MAX(sd,STDIN_FILENO);
while (1) {
rfd = rfds;
if (0 > select(maxfd+1,&rfd,NULL,NULL,NULL)) {
perror("select() failed");
exit(1);
}
(...)
}
```

Empfangen von Daten

- Sobald etwas Interessantes passiert, wird `select()` uns das sagen.
- Wir müssen feststellen, welcher der File-Deskriptoren bereit ist.
- Im Fall der Standardeingabe (STDIN) können wir mit `gets()` (oder etwas besserem) einfach die Eingabe lesen
- Wenn der File-Deskriptor des Sockets bereit ist, könnten wir `read()` verwenden, erfahren dann aber nie, wer uns was geschickt hat.
- Besser wir nutzen `recvfrom()`: Hier können wir ein `struct sockaddr_in` übergeben, in das uns `recvfrom()` reinschreibt, von wem wir etwas empfangen haben.

Ein recvfrom() für unser Programm:

```
while (1) {
(...)
if (FD_ISSET(sd,&rfd)) {
if (0 > (len=recvfrom(sd,buffer,BUFFLEN-1,0,(struct sockaddr *)&from,&slen))) {
perror("recvfrom() failed");
exit(1);
}
fprintf(stdout,"% s:%d >> %s\n",inet_ntoa(from.sin_addr),ntohs(from.sin_port),buffer);
}
(...)
}
```

Senden von Daten

- Um Daten zu Senden, müssen wir `sendto()` nutzen.
- Diesem muss man ein `struct sockaddr_in` übergeben, in dem steht, wer der Empfänger sein soll.
- Ein einfaches `write()` funktioniert nicht, da das Betriebssystem dann nicht weiß, an wen es die Daten senden soll.

Ein sendto() für unser Programm:

```
while (1) {
(...)
if (FD_ISSET(STDIN_FILENO,&rfd)) {
if (NULL == (s=fgets(buffer,BUFFLEN,stdin)))
continue;
if (0 > (len=sendto(sd,buffer,strlen(buffer),0,(struct sockaddr *)&remote,
sizeof(remote)))) {
perror("sendto() failed");
exit(1);
}
}
(...)
}
```

4.4. Verbindungsorientierte Übertragung

4.4.0.8. Verbindungsorientierte Übertragung

Grundlegende Idee: Linear durchnummeriere Segmente mittels Sequenznummern im Protkollheader Sequenznummern ermöglichen insbesondere

- Bestätigung erfolgreich übertragener Segmente,
- Identifikation fehlender Segmente,
- erneutes Anfordern fehlender Segmente und
- Zusammensetzen der Segmente in der richtigen Reihenfolge.

Probleme: Sender und Empfänger müssen

- sich zunächst synchronisieren (Austausch der initialen Sequenznummern) und
- Zustand halten (aktuelle Sequenznummer, bereits bestätigte Segmente, ...).

Grundlegende Idee: Linear durchnummeriere Segmente mittels Sequenznummern im Protkollheader Sequenznummern ermöglichen insbesondere

- Bestätigung erfolgreich übertragener Segmente,
- Identifikation fehlender Segmente,
- erneutes Anfordern fehlender Segmente und
- Zusammensetzen der Segmente in der richtigen Reihenfolge.

Probleme: Sender und Empfänger müssen

- sich zunächst synchronisieren (Austausch der initialen Sequenznummern) und
- Zustand halten (aktuelle Sequenznummer, bereits bestätigte Segmente, ...).

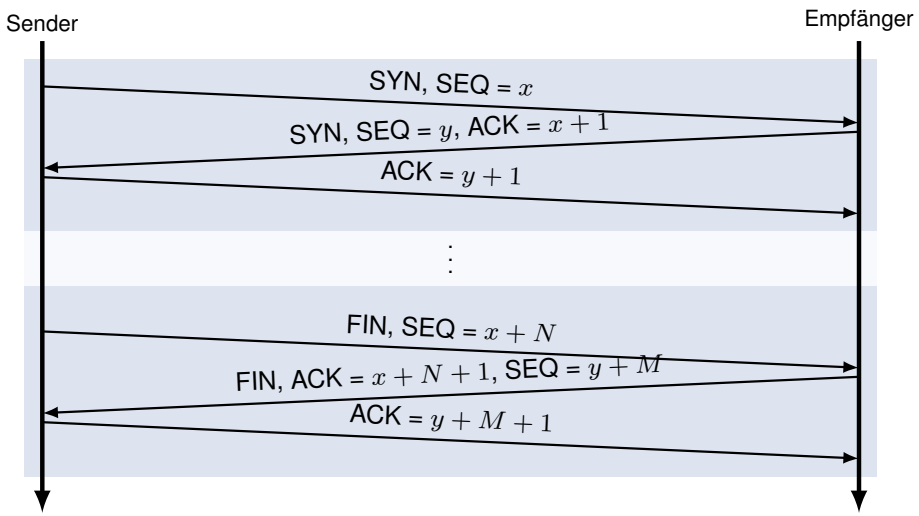
Verbindungsphasen:

1. **Verbindungsaufbau (Handshake)**
2. **Datenübertragung**
3. **Verbindungsabbau (Teardown)**

Vereinbarungen: Wir gehen zunächst davon aus,

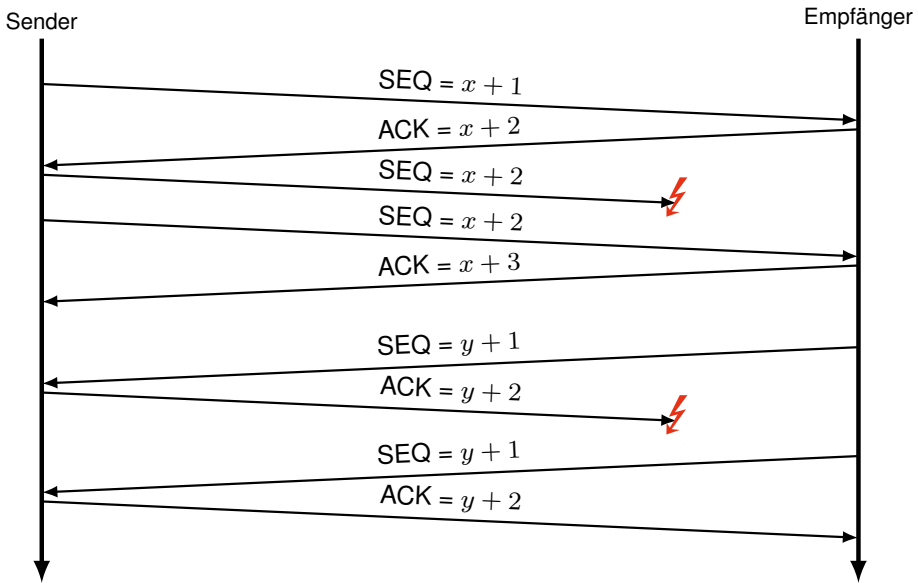
- dass stets ganze Segmente bestätigt werden und
- dass in einer Quittung das nächste erwartete Segment angegeben wird.

Beispiel: Aufbau und Abbau einer Verbindung



Diese Art des Verbidnungsaufbaues bezeichnet man als 3-Way-Handshake.

Beispiel: Übertragungsphase



4.4.1. Sliding-Window-Verfahren

Sliding-Window Verfahren Bislang:

- Im vorherigen Beispiel hat der Sender stets nur ein Segment gesendet und dann auf eine Bestätigung gewartet
- Dieses Verfahren ist ineffizient, da abhängig von der Umlaufverzögerung (Round Trip Time, RTT) zwischen Sender und Empfänger viel Bandbreite ungenutzt bleibt („Stop and Wait"-Verfahren)

Idee: Teile dem Sender mit, wie viele Segmente **nach** dem letzten bestätigten Segment auf einmal übertragen werden dürfen, ohne dass der Sender auf eine Bestätigung warten muss.
Vorteile:

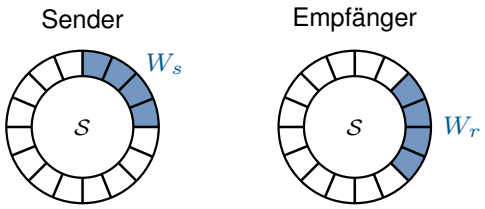
- Zeit zwischen dem Absenden eines Segments und dem Eintreffen einer Bestätigung kann effizienter genutzt werden
- Durch die Aushandlung dieser Fenstergrößen kann der Empfänger die Datenrate steuern → Flusskontrolle
- Durch algorithmische Anpassung der Fenstergröße kann die Datenrate an die verfügbare Datenrate auf dem Übertragungspfad zwischen Sender und Empfänger angepasst werden → Staukontrolle

Probleme:

- Sender und Empfänger müssen mehr Zustand halten
  (Was wurde bereits empfangem? Was wird als nächstes erwartet?)
- Der Sequenznummernraum ist endlich → Wie werden Missverständnisse verhindert?

Zur Notation:

- Sender und Empfänger haben denselben Sequenznummernraum $\mathcal{S} = \{0, 1, 2, \ldots, N-1\}$.
  Beispiel: $N = 16$:



- Sendefenster (Send Window) $W_s \subset \mathcal{S}$, $|W_s| = w_s$:
  Es dürfen w_s Segmente nach dem letzten bestätigten Segment auf einmal gesendet werden.

- Empfangsfenster (Receive Window) $W_r \subset \mathcal{S}$, $|W_r| = w_r$:
  Sequenznummern der Segmente, die als nächstes akzeptiert werden.

- Sende- und Empfangsfenster „verschieben“ und überlappen sich während des Datenaustauschs.

Vereinbarungen:

- Eine Bestätigung ACK $= m + 1$ bestätigt alle Segmente mit SEQ $\leq m$. Dies wird als kumulative Bestätigung bezeichnet.

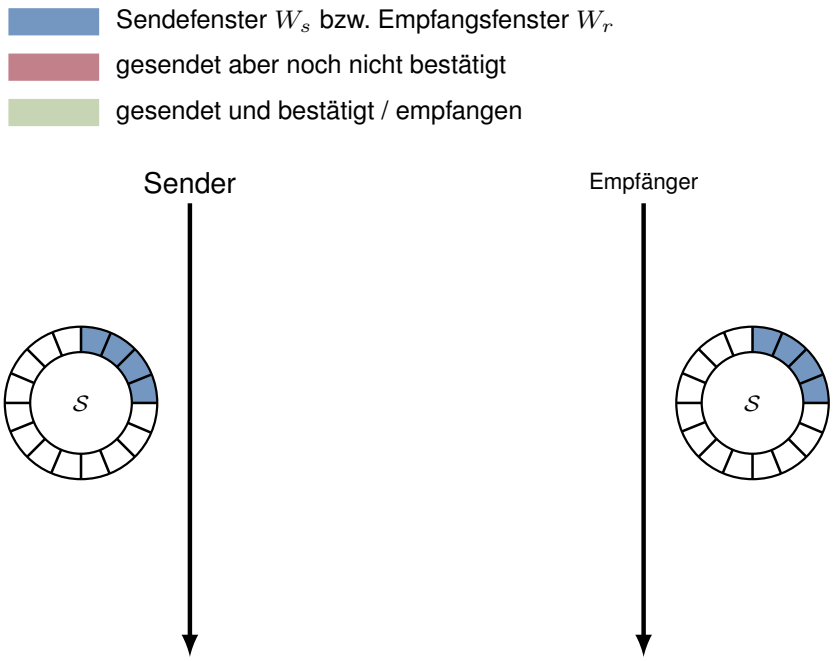
- Gewöhnlich löst jedes erfolgreich empfangene Segment das Senden einer Bestätigung aus, wobei stets das nächste erwartete Segment bestätigt wird. Dies wird als Forward Acknowledgement bezeichnet.

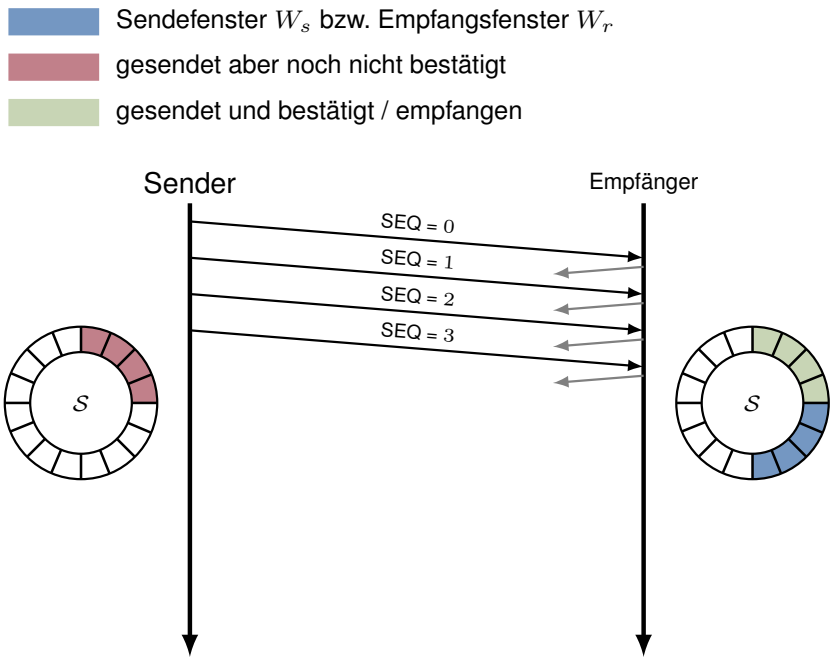
Wichtig:

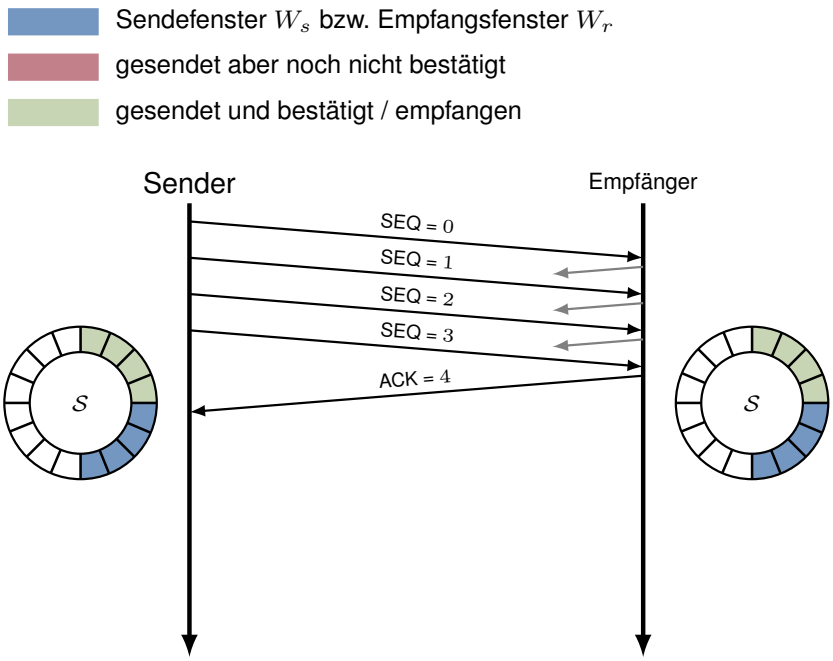
- In den folgenden Grafiken sind die meisten Bestätigungen zwecks Übersichtlichkeit nur angedeutet (graue Pfeile).

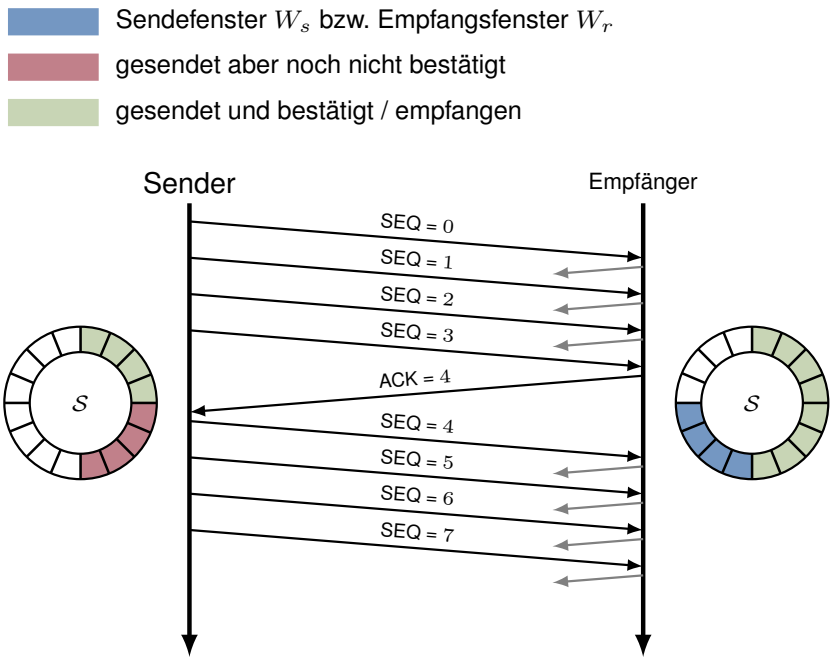
- Die Auswirkungen auf Sende- und Empfangsfenster beziehen sich nur auf den Erhalt der schwarz eingezeichneten Bestätigungen.

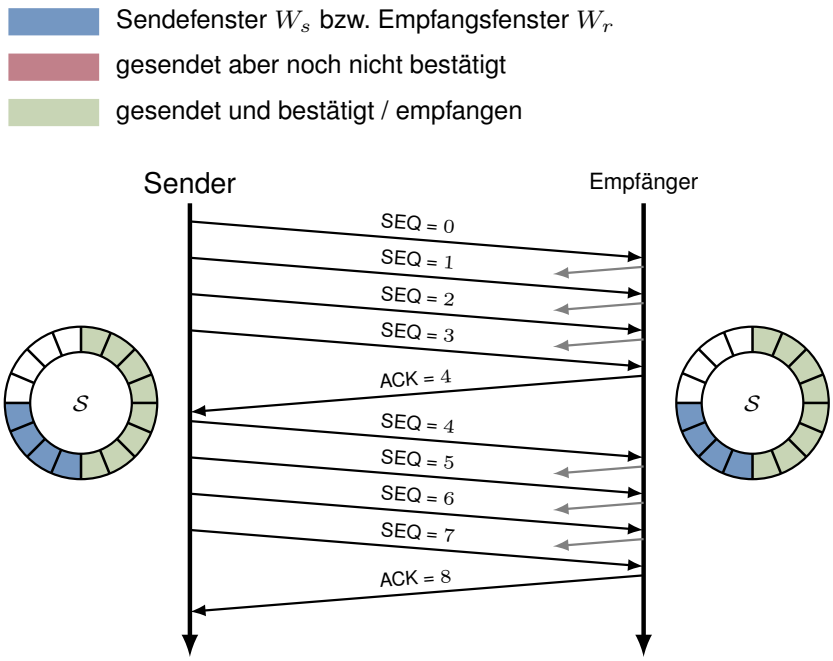
- Dies ist äquivalent zur Annahme, dass die angedeuteten Bestätigungen verloren gehen.


Sendefenster W_s bzw. Empfangsfenster W_r
gesendet aber noch nicht bestätigt
gesendet und bestätigt / empfangen
Sender
Empfänger
S
S


Sendefenster W_s bzw. Empfangsfenster W_r
gesendet aber noch nicht bestätigt
gesendet und bestätigt / empfangen
Sender
Empfänger
SEQ = 0
SEQ = 1
SEQ = 2
SEQ = 3
S
S


Sendefenster W_s bzw. Empfangsfenster W_r
gesendet aber noch nicht bestätigt
gesendet und bestätigt / empfangen
Sender
Empfänger
SEQ = 0
SEQ = 1
SEQ = 2
SEQ = 3
ACK = 4
S
S


Sendefenster W_s bzw. Empfangsfenster W_r
gesendet aber noch nicht bestätigt
gesendet und bestätigt / empfangen
Sender
Empfänger
SEQ = 0
SEQ = 1
SEQ = 2
SEQ = 3
ACK = 4
SEQ = 4
SEQ = 5
SEQ = 6
SEQ = 7
S
S



Neues Problem: Wie wird jetzt mit Segmentverlusten umgegangen? **Zwei Möglichkeiten:**

1. **Go-Back-N**
   - Akzeptiere stets nur die nächste erwartete Sequenznummer
   - Alle anderen Segmente werden verworfen
2. **Selective-Repeat**
   - Akzeptiere alle Sequenznummern, die in das aktuelle Empfangsfenster fallen
   - Diese müssen gepuffert werden, bis fehlende Segmente erneut übertragen wurden

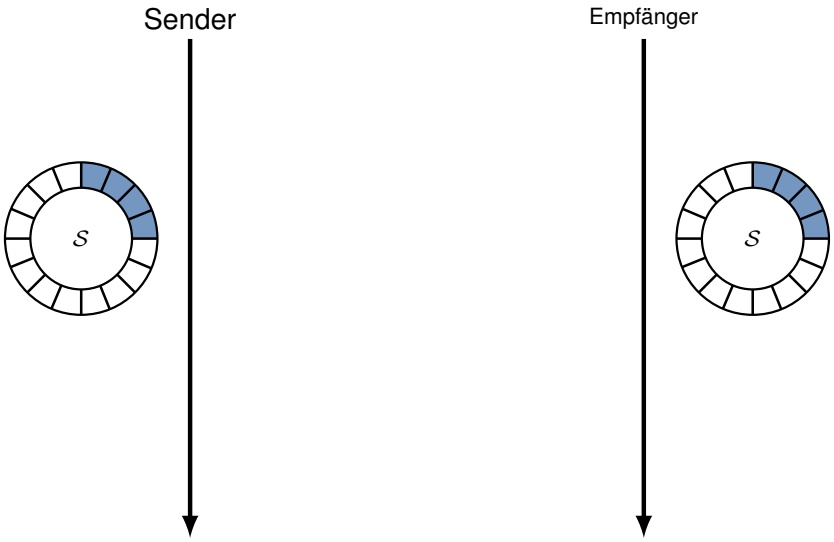
Wichtig:

- In beiden Fällen muss der Sequenznummernraum so gewählt werden, dass wiederholte Segmente eindeutig von neuen Segmenten unterschieden werden können.
- Andernfalls würde es zu Verwechslungen kommen
  → Auslieferung von Duplikaten an höhere Schichten, keine korrekte Reihenfolge

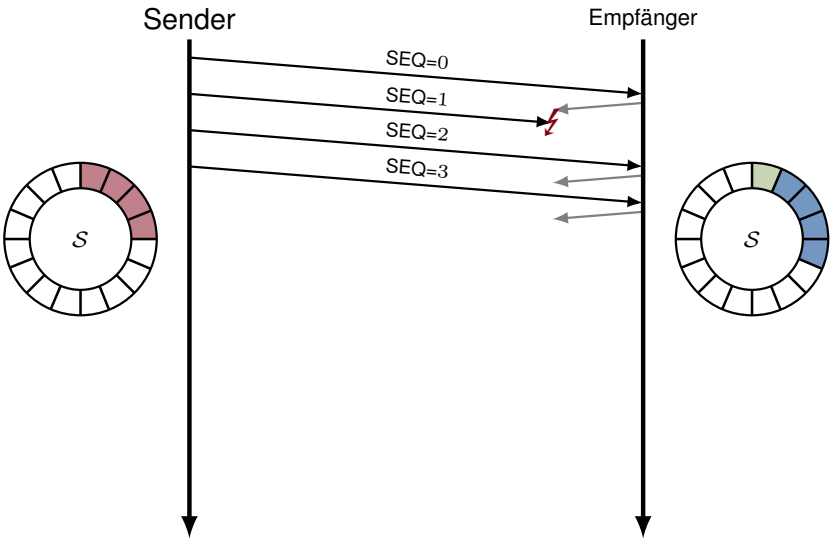
Frage: (s. Übung) Wie groß darf das Sendefenster W_s in Abhängigkeit des Sequenznummernraums $\mathcal{S}$ höchstens gewählt werden, so dass die Verfahren funktionieren?

4.4.1.1. Go-Back-N

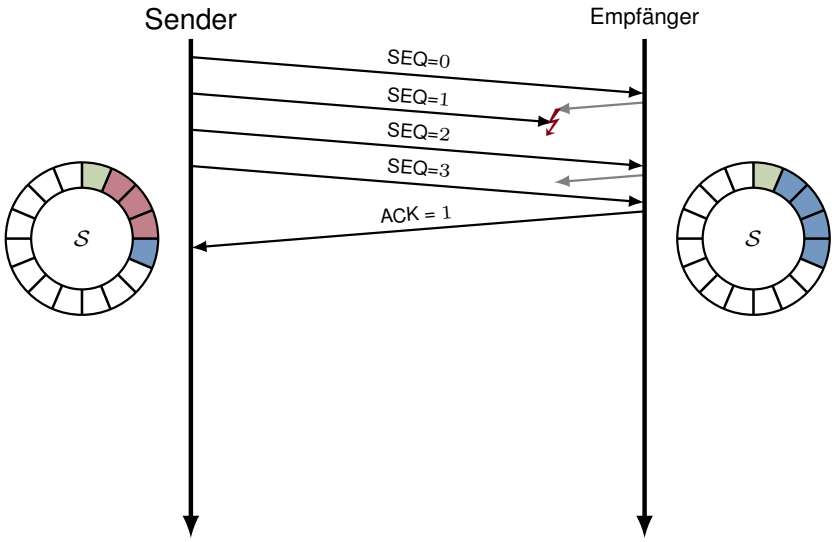
Go-Back-N: $N = 16$, $w_s = 4$, $w_r = 4$



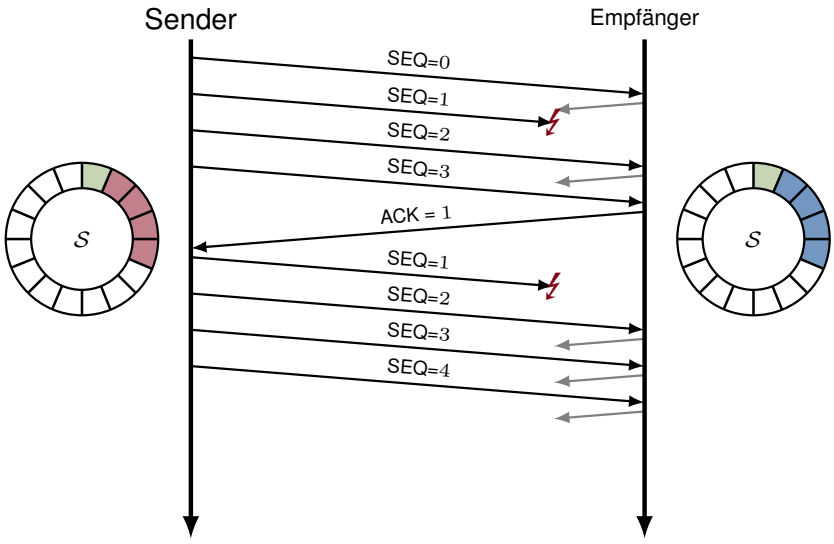
Go-Back-N: $N = 16$, $w_s = 4$, $w_r = 4$



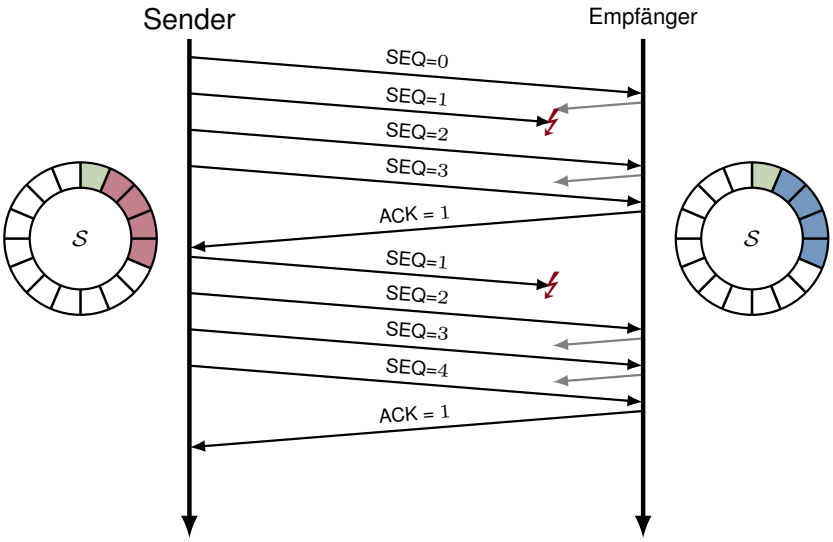
Go-Back-N: $N = 16$, $w_s = 4$, $w_r = 4$



Go-Back-N: $N = 16$, $w_s = 4$, $w_r = 4$



Go-Back-N: $N = 16$, $w_s = 4$, $w_r = 4$



Anmerkungen zu Go-Back-N

- Da der Empfänger stets nur das nächste erwartete Segment akzeptiert, reicht ein Empfangsfenster der Größe $w_r = 1$ prinzipiell aus. Unabhängig davon muss für praktische Implementierungen ein ausreichend großer Empfangspuffer verfügbar sein.

- Bei einem Sequenznummernraum der Kardinalität N muss für das Sendefenster stets gelten:

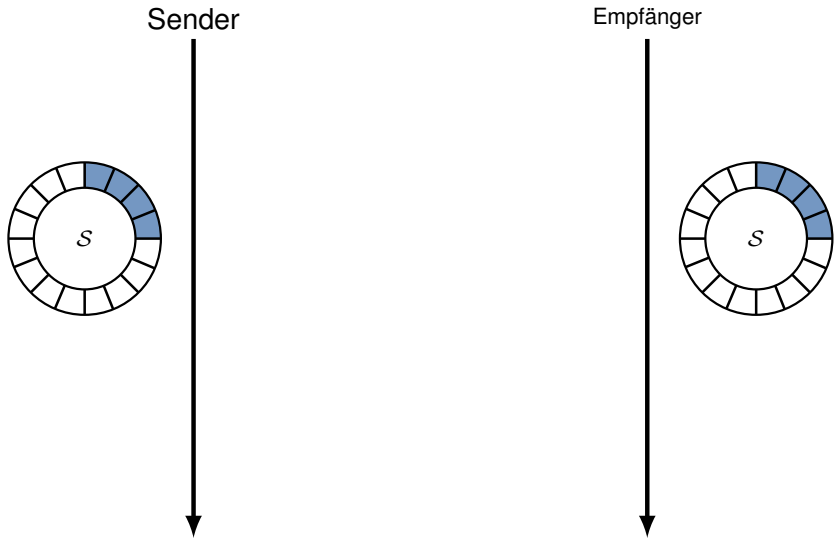
  $$w_s \leq N - 1.$$

  Andernfalls kann es zu Verwechslungen kommen (s. Übung).

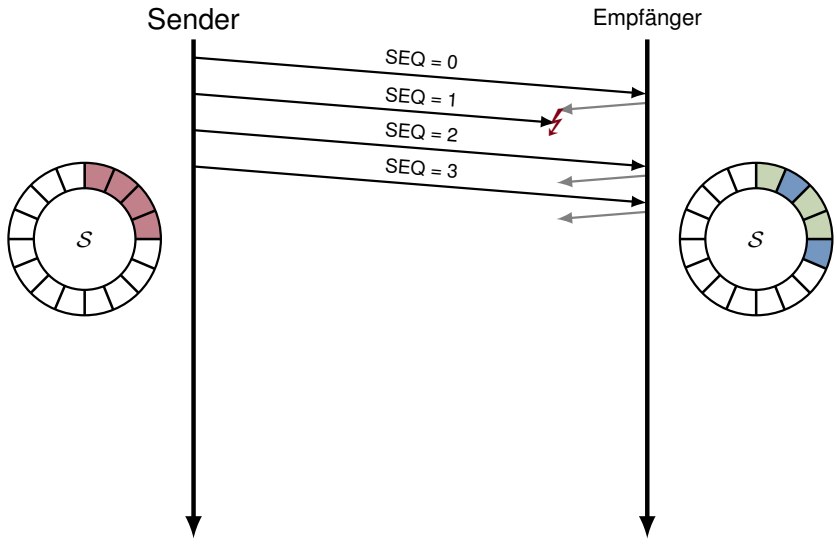
- Das Verwerfen erfolgreich übertragener aber nicht in der erwarteten Reihenfolge eintreffender Segmente macht das Verfahren einfach zu implementieren aber weniger effizient.

4.4.1.2. Selective Repeat

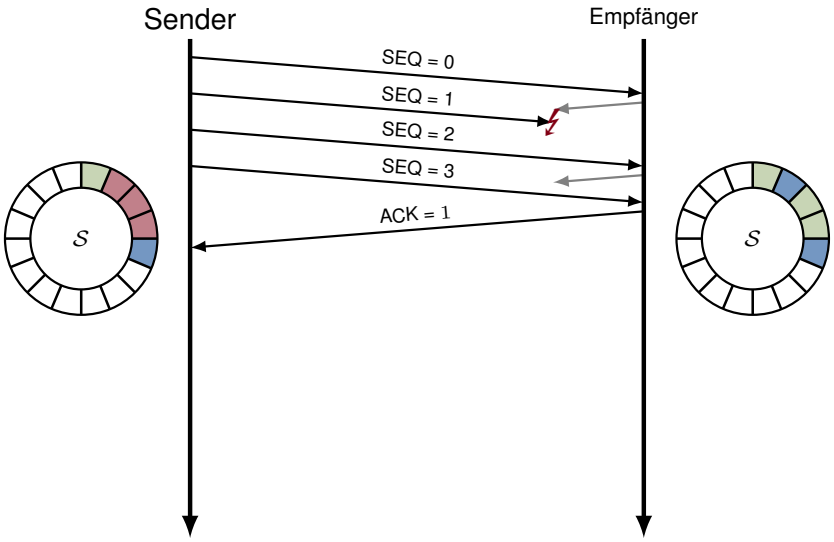
Selective Repeat: $N = 16$, $w_s = 4$, $w_r = 4$



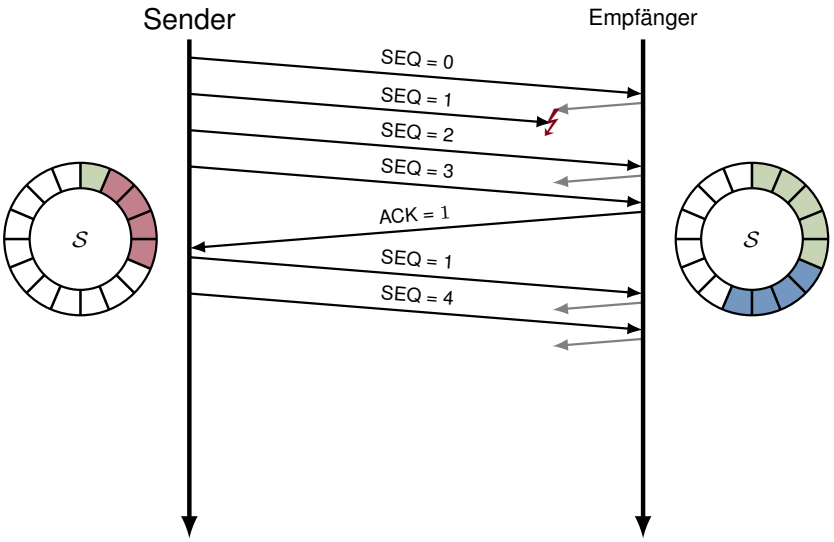
Selective Repeat: $N = 16$, $w_s = 4$, $w_r = 4$



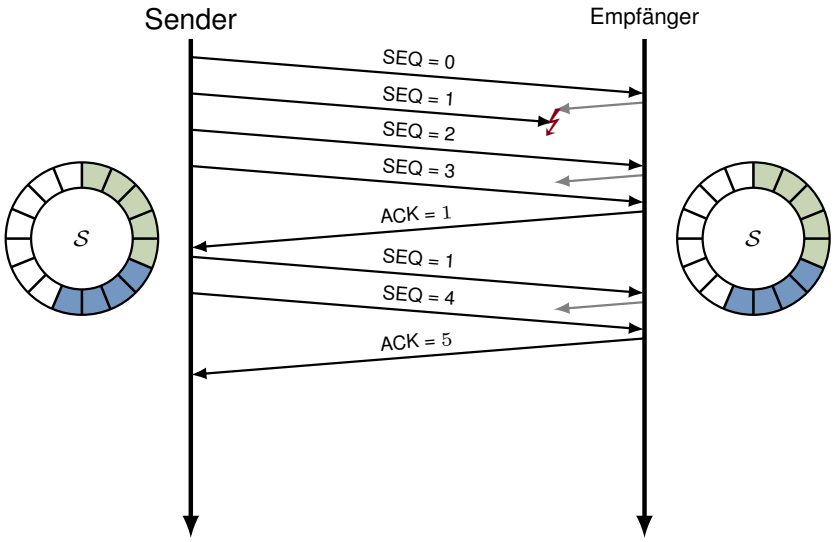
Selective Repeat: $N = 16$, $w_s = 4$, $w_r = 4$



Selective Repeat: $N = 16$, $w_s = 4$, $w_r = 4$



Selective Repeat: $N = 16$, $w_s = 4$, $w_r = 4$



Anmerkungen zu Selective Repeat

- Wählt man $w_r = 1$ und w_s unabhängig von w_r, so degeneriert Selective Repeat zu Go-Back-N.
- Bei einem Sequenznummernraum der Kardinalität N muss für das Sendefenster stets gelten:

$$w_s \leq \left\lfloor \frac{N}{2} \right\rfloor .$$

  Andernfalls kann es zu Verwechslungen kommen (s. Übung).

Allgemeine Anmerkungen

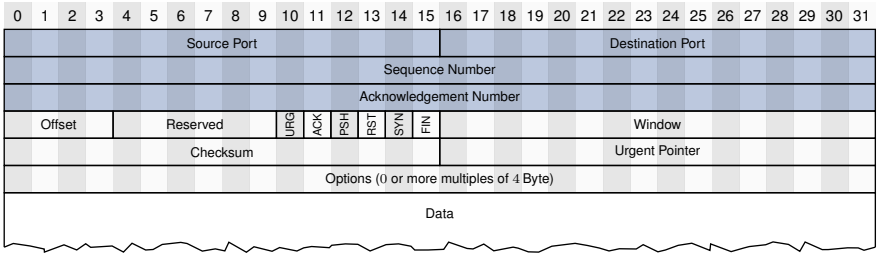
- Bei einer Umsetzung dieser Konzepte benötigt insbesondere der Empfänger einen Empfangspuffer, dessen Größe an die Sende- und Empfangsfenster angepasst ist.
- Für praktische Anwendungen werden die Größen von W_s und W_r dynamisch angepasst (siehe Case Study zu TCP), wodurch Algorithmen zur Staukontrolle und Flusskontrolle auf Schicht 4 ermöglicht werden.

4.4.2. Transmission Control Protocol (TCP)

Transmission Control Protocol (TCP) Das Transmission Control Protocol (TCP) ist das dominierende Transportprotokoll im Internet. Es bietet

- gesicherte / stromorientierte Übertragung sowie
- Mechanismen für Fluss- und Staukontrolle.

TCP-Header:

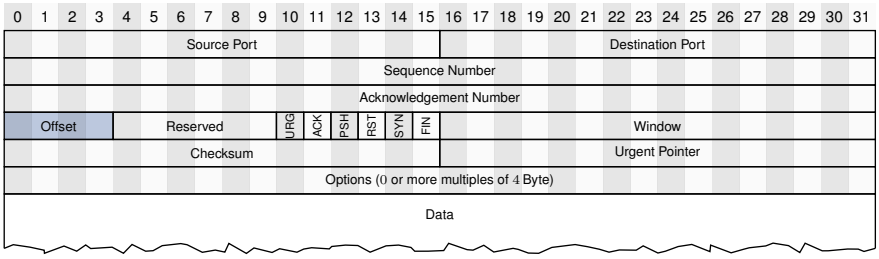


- Quell- und Zielport werden analog zu UDP verwendet.
- Sequenz- und Bestätigungsnummer dienen der gesicherten Übertragung. Es werden bei TCP nicht ganze Segmente sondern einzelne Byte bestätigt (stromorientierte Übertragung).

Transmission Control Protocol (TCP) Das Transmission Control Protocol (TCP) ist das dominierende Transportprotokoll im Internet. Es bietet

- gesicherte / stromorientierte Übertragung sowie
- Mechanismen für Fluss- und Staukontrolle.

TCP-Header:



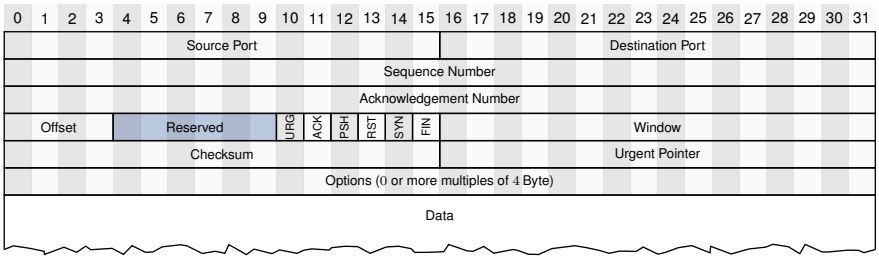
(Data) Offset

- Gibt die Länge des TCP-Headers in Vielfachen von 4 Byte an.
- Der TCP-Header hat variable Länge (Optionen, vgl. IP-Header).

Transmission Control Protocol (TCP) Das Transmission Control Protocol (TCP) ist das dominierende Transportprotokoll im Internet. Es bietet

- gesicherte / stromorientierte Übertragung sowie
- Mechanismen für Fluss- und Staukontrolle.

TCP-Header:



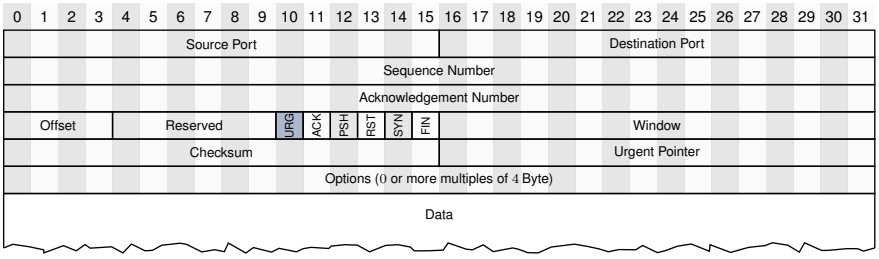
Reserved

- In bisherigen TCP-Versionen keine Verwendung. Muss auf 0 gesetzt werden, so dass zukünftige TCP-Versionen bei Bedarf das Feld nutzen können.

Transmission Control Protocol (TCP) Das Transmission Control Protocol (TCP) ist das dominierende Transportprotokoll im Internet. Es bietet

- gesicherte / stromorientierte Übertragung sowie
- Mechanismen für Fluss- und Staukontrolle.

TCP-Header:



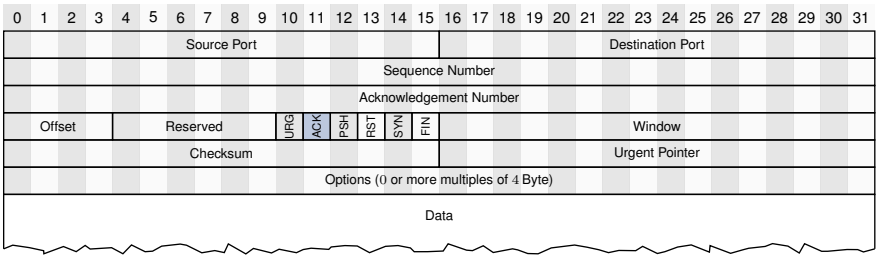
Flag URG („urgent“) (selten verwendet)

- Ist das Flag gesetzt, werden die Daten im aktuellen TCP-Segment beginnend mit dem ersten Byte bis zu der Stelle, an die das Feld Urgent Pointer zeigt, sofort an höhere Schichten weitergeleitet.

Transmission Control Protocol (TCP) Das Transmission Control Protocol (TCP) ist das dominierende Transportprotokoll im Internet. Es bietet

- gesicherte / stromorientierte Übertragung sowie
- Mechanismen für Fluss- und Staukontrolle.

TCP-Header:



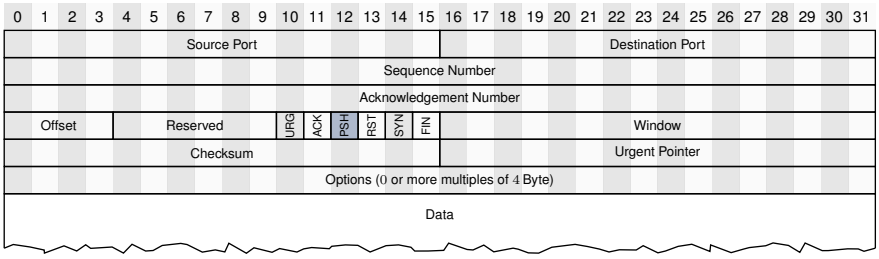
Flag ACK („acknowledgement“)

- Ist das Flag gesetzt, handelt es sich um eine Empfangsbestätigung.
- Bestätigungen können bei TCP auch „huckepack“ (engl. piggy backing) übertragen werden, d. h. es werden gleichzeitig Nutzdaten von A nach B übertragen und ein zuvor von B nach A gesendetes Segment bestätigt.
- Die Acknowledgement-Number gibt bei TCP stets das nächste erwartete Byte an.

Transmission Control Protocol (TCP) Das Transmission Control Protocol (TCP) ist das dominierende Transportprotokoll im Internet. Es bietet

- gesicherte / stromorientierte Übertragung sowie
- Mechanismen für Fluss- und Staukontrolle.

TCP-Header:



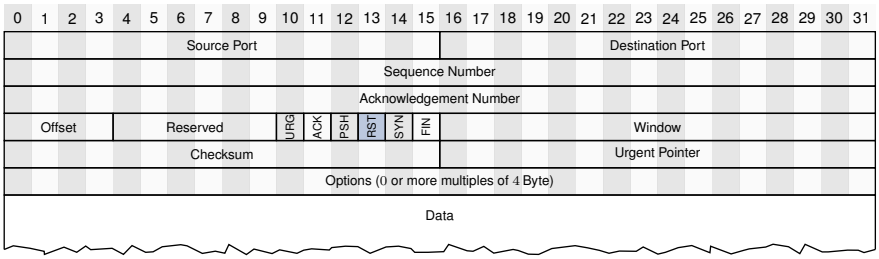
Flag PSH („push“)

- Ist das Flag gesetzt, werden sende- und empfangsseitige Puffer des TCP-Stacks umgangen.
- Sinnvoll für interaktive Anwendungen (z. B. Telnet-Verbindungen).

Transmission Control Protocol (TCP) Das Transmission Control Protocol (TCP) ist das dominierende Transportprotokoll im Internet. Es bietet

- gesicherte / stromorientierte Übertragung sowie
- Mechanismen für Fluss- und Staukontrolle.

TCP-Header:



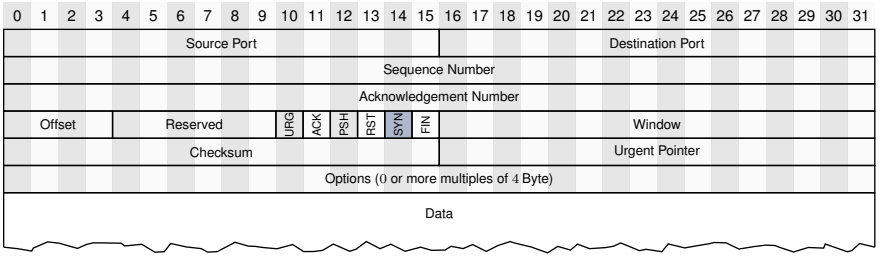
Flag RST („reset“)

- Dient dem Abbruch einer TCP-Verbindung ohne ordnungsgemäßen Verbindungsabbau.

Transmission Control Protocol (TCP) Das Transmission Control Protocol (TCP) ist das dominierende Transportprotokoll im Internet. Es bietet

- gesicherte / stromorientierte Übertragung sowie
- Mechanismen für Fluss- und Staukontrolle.

TCP-Header:



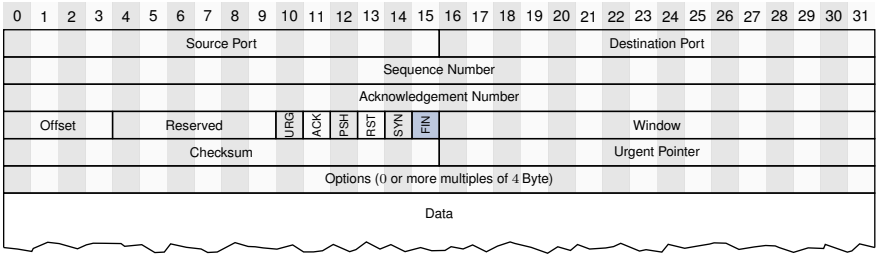
Flag SYN („synchronization“)

- Ist das Flag gesetzt, handelt es sich um ein Segment, welches zum Verbindungsaufbau gehört (initialer Austausch von Sequenznummern).
- Ein gesetztes SYN-Flag inkrementiert Sequenz- und Bestätigungsnummern um 1 obwohl keine Nutzdaten transportiert werden.

Transmission Control Protocol (TCP) Das Transmission Control Protocol (TCP) ist das dominierende Transportprotokoll im Internet. Es bietet

- gesicherte / stromorientierte Übertragung sowie
- Mechanismen für Fluss- und Staukontrolle.

TCP-Header:



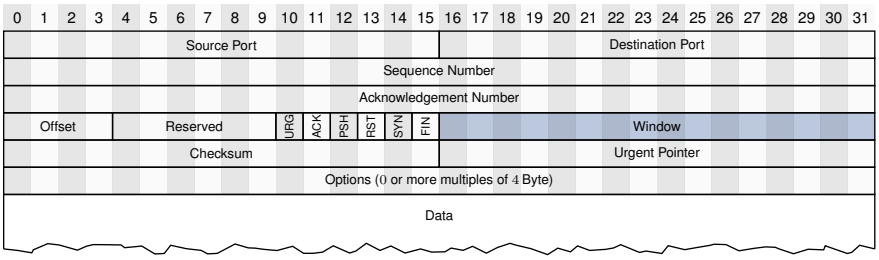
Flag FIN („finish“)

- Ist das Flag gesetzt, handelt es sich um ein Segment, welches zum Verbindungsabbau gehört.
- Ein gesetztes FIN-Flag inkrementiert Sequenz- und Bestätigungsnummern um 1 obwohl keine Nutzdaten transportiert werden.

Transmission Control Protocol (TCP) Das Transmission Control Protocol (TCP) ist das dominierende Transportprotokoll im Internet. Es bietet

- gesicherte / stromorientierte Übertragung sowie
- Mechanismen für Fluss- und Staukontrolle.

TCP-Header:



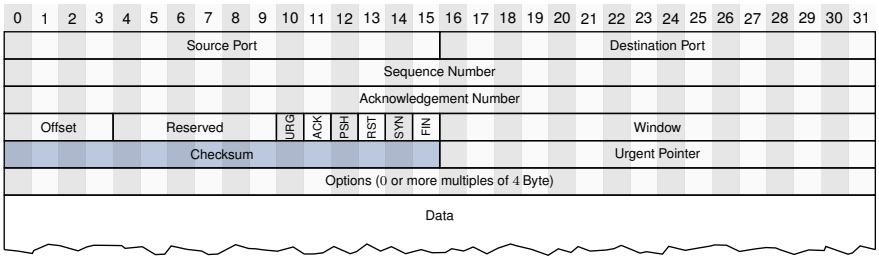
Receive Window

- Größe des aktuellen Empfangsfensters W_r in Byte.
- Ermöglicht es dem Sender, die Datenrate des Senders zu drosseln.

Transmission Control Protocol (TCP) Das Transmission Control Protocol (TCP) ist das dominierende Transportprotokoll im Internet. Es bietet

- gesicherte / stromorientierte Übertragung sowie
- Mechanismen für Fluss- und Staukontrolle.

TCP-Header:



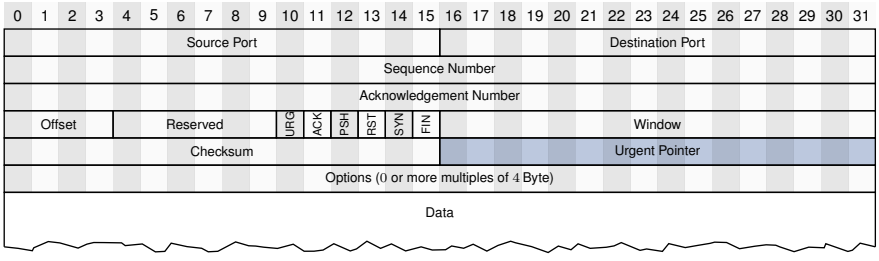
Checksum

- Checksumme über Header und Daten.
- Wie bei UDP wird zur Berechnung ein Pseudo-Header verwendet.

Transmission Control Protocol (TCP) Das Transmission Control Protocol (TCP) ist das dominierende Transportprotokoll im Internet. Es bietet

- gesicherte / stromorientierte Übertragung sowie
- Mechanismen für Fluss- und Staukontrolle.

TCP-Header:



Urgent Pointer (selten verwendet)

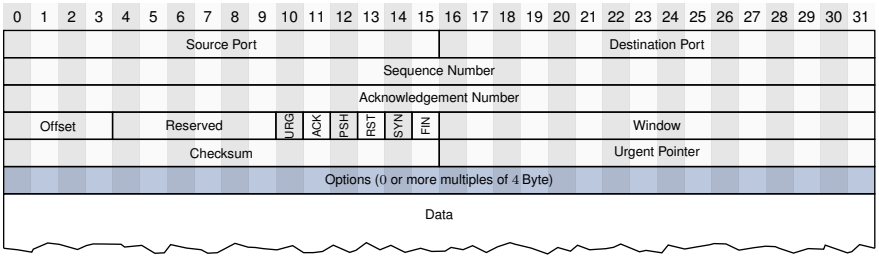
- Gibt das Ende der „Urgent-Daten“ an, welche unmittelbar nach dem Header beginnen und bei gesetztem URG-Flag sofort an höhere Schichten weitergereicht werden sollen.

Transmission Control Protocol (TCP) Das Transmission Control Protocol (TCP) ist das dominierende Transportprotokoll im Internet. Es bietet

- gesicherte / stromorientierte Übertragung sowie

- Mechanismen für Fluss- und Staukontrolle.

TCP-Header:



Options

- Zusätzliche Optionen, z. B. Window Scaling (s. Übung), selektive Bestätigungen oder Angabe der Maximum Segment Size (MSS).

Anmerkungen zur MSS

- Die MSS gibt die maximale Größe eines TCP-Segments (Nutzdaten ohne TCP-Header) an.
- Zum Vergleich gibt die MTU (Maximum Transfer Unit) die maximale Größe der Nutzdaten aus Sicht von Schicht 2 an (alles einschließlich des IP-Headers).
- In der Praxis sollte die MSS so gewählt werden, dass keine IP-Fragmentierung beim Senden notwendig ist.

Beispiele:

- MSS bei FastEthernet
  - MTU beträgt 1500 B
  - Davon entfallen 20 B auf den IP-Header und weitere 20 B auf den TCP-Header (sofern keine Optionen verwendet werden)
  - Die sinnvolle MSS beträgt demnach 1460 B.
- DSL-Verbindungen
  - Zwischen Ethernet- und IP-Header wird ein 8 B PPPoE-Header eingefügt
  - Demzufolge sollte die MSS auf 1452 B reduziert werden
- VPN-Verbindungen
  - Abhängig vom eingesetzten Verschlüsselungsverfahren sind weitere Header notwendig
  - Die sinnvolle MSS ist hier nicht immer offensichtlich

Fluss- und Staukontrolle bei TCP

TCP-Flusskontrolle:

Ziel der Flusskontrolle ist es, Überlastsituationen beim Empfänger zu vermeiden. Dies wird erreicht, indem der Empfänger eine Maximalgröße für das Sendefenster des Senders vorgibt.

- Empfänger teilt dem Sender über das Feld Receive Window im TCP-Header die aktuelle Größe des Empfangsfensters W_r mit.
- Der Sender interpretiert diesen Wert als die maximale Anzahl an Byte, die ohne Abwarten einer Bestätigung übertragen werden dürfen.
- Durch Herabsetzen des Wertes kann die Übertragungsrate des Senders gedrosselt werden, z. B. wenn sich der Empfangspuffer des Empfängers füllt.

TCP-Staukontrolle:

Ziel der **Staukontrolle** ist es, Überlastsituationen im Netz zu vermeiden. Dazu muss der Sender Engpässe im Netz erkennen und die Größe des Sendefensters entsprechend anpassen.

Zu diesem Zweck wird beim Sender zusätzlich ein Staukontrollfenster (engl. Congestion Window) W_c eingeführt, dessen Größe wir mit w_c bezeichnen:

- W_c wird vergrößert, solange Daten verlustfrei übertragen werden
- W_c wird verkleinert, wenn Verluste auftreten
- Für das tatsächliche Sendefenster gilt stets $w_s = \min\{w_c, w_r\}$

TCP-Staukontrolle Man unterscheidet bei TCP zwischen zwei Phasen der Staukontrolle:

1. **Slow-Start:**
   Für jedes bestätigte Segment wird W_c um eine MSS vergrößert. Dies führt zu **exponentiellem Wachstum** des Staukontrollfensters bis ein Schwellwert (engl. **Congestion Threshold**) erreicht ist. Danach wird mit der Congestion-Avoidance-Phase fortgefahren.

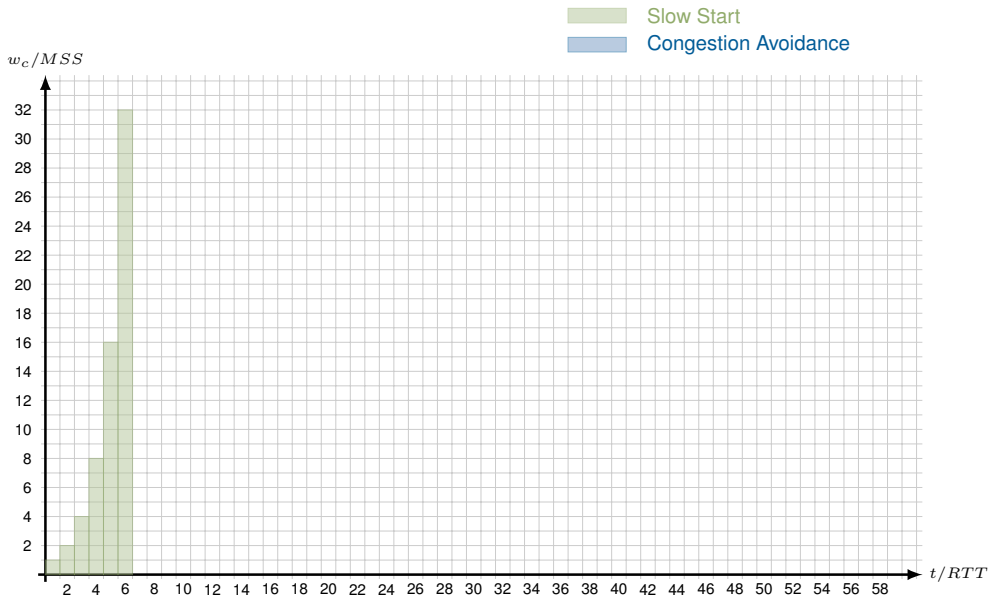
2. **Congestion Avoidance:**
   Für jedes bestätige Segment wird W_c lediglich um $(1/w_c) \cdot \text{MSS}$ vergrößert, d. h. nach Bestätigung eines vollständigen Staukontrollfensters um genau eine MSS. Dies führt zu linearem Wachstum des Staukontrollfensters.

Es gibt mehrere Varianten von TCP, welche sich hinsichtlich ihres Verhaltens beim Auftreten von Segmentverlusten unterscheiden. Im Folgenden beziehen wir uns auf „TCP-Reno“:

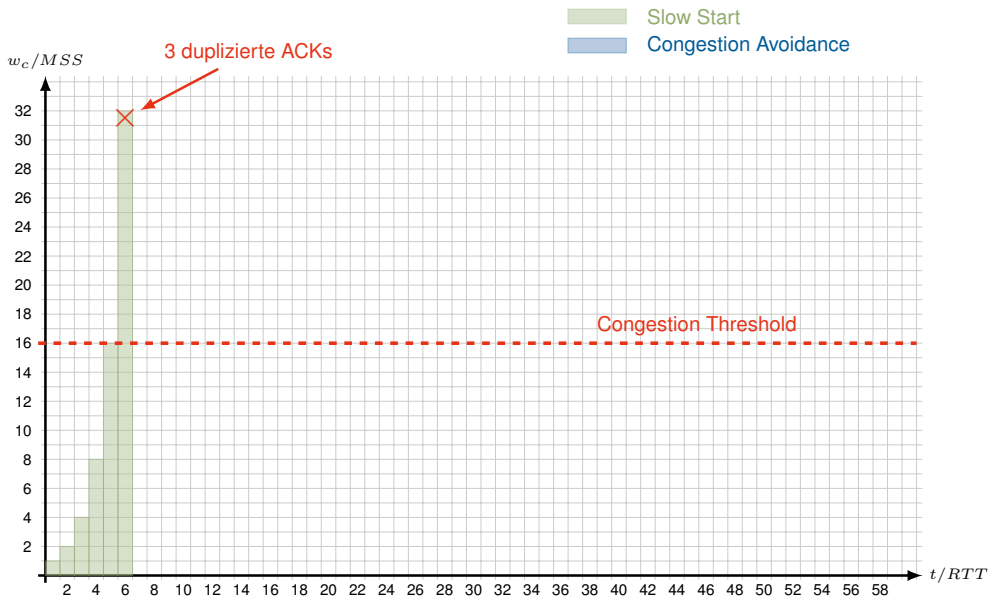
1. **3 duplizierte Bestätigungen (Duplicate ACKs)**
   - Setze den Schwellwert für die Stauvermeidung auf die $w_c/2$
   - Reduziere W_c auf die Größe dieses Schwellwerts
   - Beginne mit der Stauvermeidungsphase
2. **Timeout**
   - Setze den Schwellwert für die Stauvermeidung auf die $w_c/2$
   - Setze $w_c = 1$
   - Beginne mit einem neuen Slow-Start

- Der Vorgänger TCP-Tahoe unterscheidet nicht zwischen diesen beiden Fällen und führt immer Fall 2 aus.
- Es gibt eine Reihe weiterer TCP-Versionen, die sich insbesondere hinsichtlich ihres Staukontrollverhaltens unterscheiden.
- Grundsätzlich sind alle TCP-Versionen kompatibel zueinander, allerdings können sich die unterschiedlichen Staukontrollverfahren gegenseitig nachteilig beeinflussen.

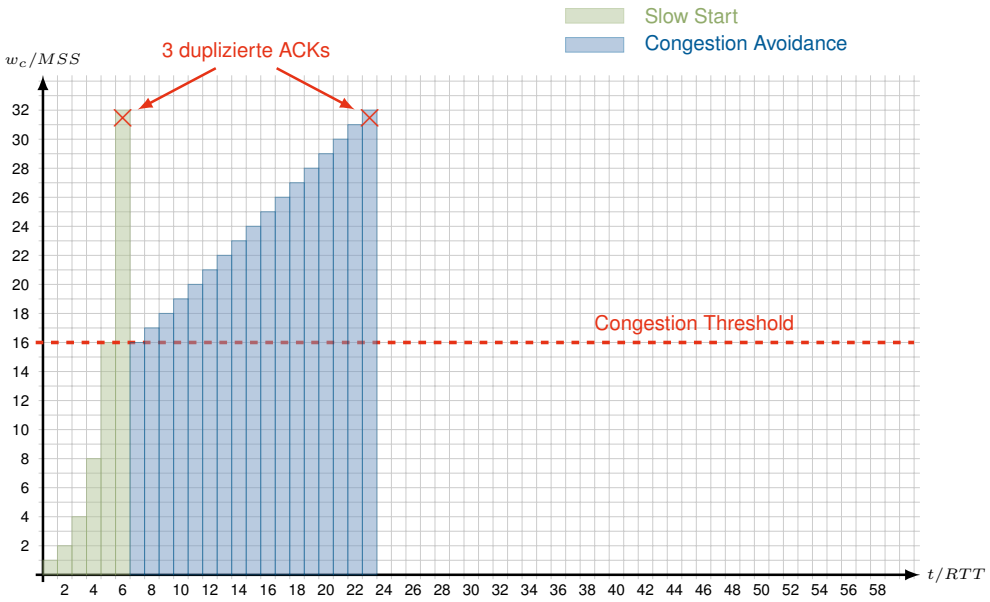
Beispiel: TCP-Reno



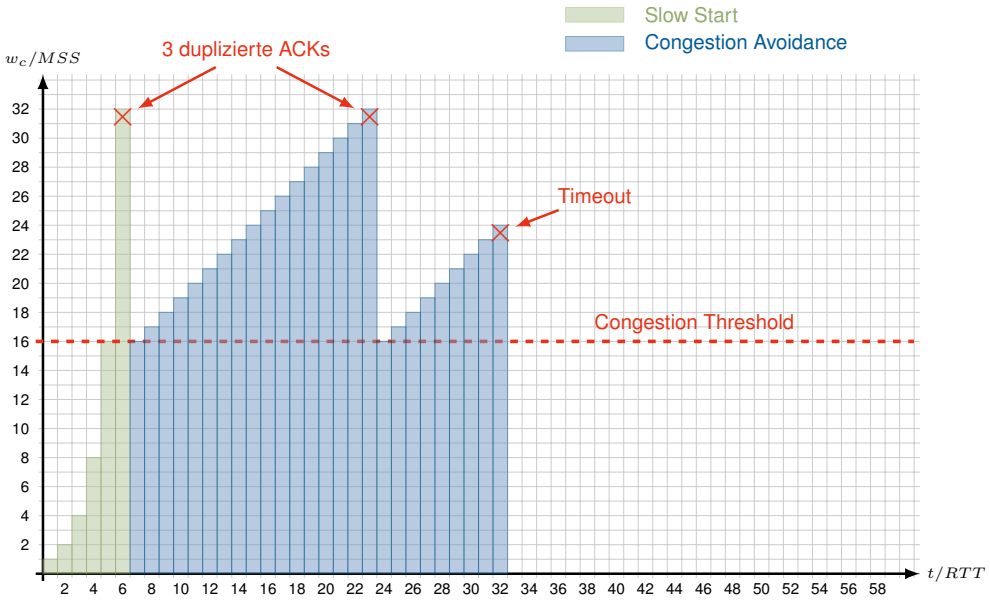
Beispiel: TCP-Reno



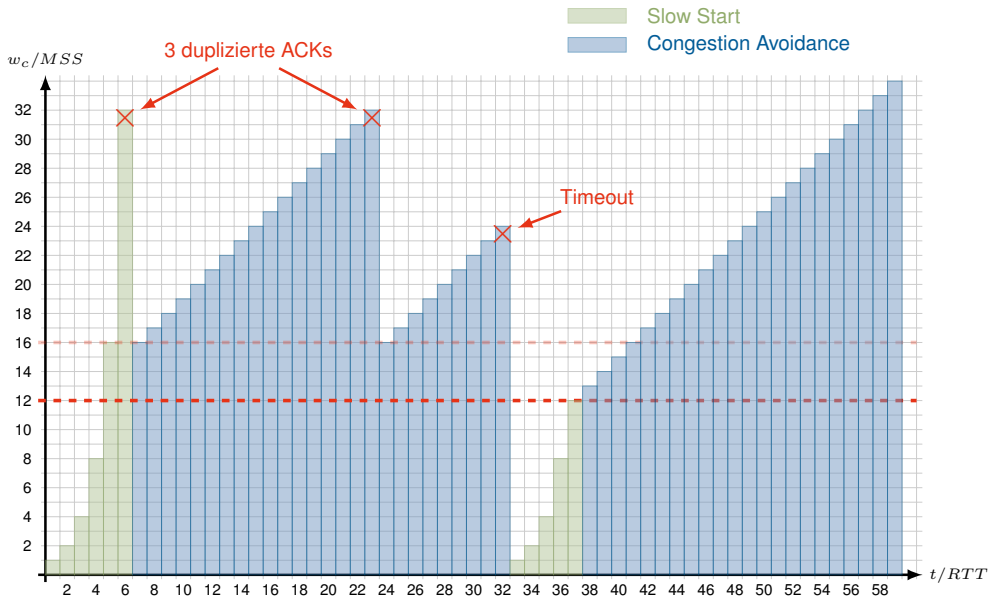
Beispiel: TCP-Reno



Beispiel: TCP-Reno



Beispiel: TCP-Reno



Anmerkungen Obwohl TCP gesicherte Verbindungen ermöglicht, dient es nicht der Kompensierung eines unzuverlässigen Data Link Layers!

- TCP interpretiert von Verlust von Paketen (Daten und Bestätigungen) stets als eine Folge einer Überlastsituation.
- In der Folge reduziert TCP die Datenrate.
- Handelt es sich bei den Paketverlusten jedoch um die Folge von Bitfehlern, so wird die Datenrate unnötiger Weise gedrosselt.
- Durch die ständige Halbierung der Datenrate oder neue Slow-Starts kann das Sendefenster nicht mehr auf sinnvolle Größen anwachsen.
- In der Praxis ist TCP bereits mit 1 % Paketverlust, der nicht auf Überlast zurückzuführen ist, bereits überfordert!

:

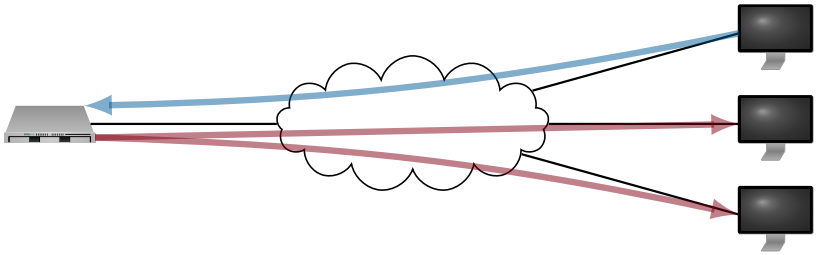
⇒ Die Schichten 1 – 3 müssen eine für TCP „ausreichend geringe“ Paketfehlerrate bereitstellen

- In der Praxis bedeutet dies, dass Verlustwahrscheinlichkeiten in der Größenordnung von 10^{-3} und niedriger notwendig bzw. anzustreben sind.
- Bei Bedarf müssen zusätzliche Bestätigungsverfahren auf Schicht 2 zum Einsatz kommen, um dies zu gewährleisten (z. B. IEEE 802.11).

Case Study: TCP-Relay-Chat Was wir wollen:

- Einen Server, welcher N Clientverbindungen gleichzeitig unterstützt
- Sendet ein Client eine Nachricht an den Server, soll diese an alle anderen Clients weitergeleitet werden
- Dies entspricht einem Chatroom

- Server und Client sind nun zwei unterschiedliche Programme



Welche Sprache?

- C natürlich (;

Der Server

- Sieht ähnlich aus wie unser UDP-Chat ...

```
if (0 > (sd=socket(AF_INET,SOCK_STREAM,IPPROTO_TCP))) {
perror("socket() failed");
exit(1);
}
local.sin_family      = AF_INET;
local.sin_port        = htons(local_port);
local.sin_addr.s_addr = INADDR_ANY;
if (0 > bind(sd,(struct sockaddr *)&local,sizeof(local))) {
perror("bind() failed");
exit(1);
}
if (0 > listen(sd,MAXCLIENTS)) {
perror("listen() failed");
exit(1);
}
```

Neu sind

- der Socket-Typ `SOCK_STREAM` und das Protokoll `IPPROTO_TCP` sowie
- der Aufruf von `listen()`, welcher den Socket als passiv markiert.

Letzteres bedeutet, dass über diesen Socket keine Daten versendet oder empfangen werden sondern stattdessen eingehende Verbindungen auf diesem Socket erwartet werden.
Verbindet sich ein Client, muss die Verbindung akzeptiert werden:

```
while (1) {
rfd = rfds;
if (0 > select(maxfd+1,&rfd,NULL,NULL,NULL)) {
perror(select() failed");
exit(1);
}
if (FD_ISSET(sd,&rfd)) {
if (0 > (csd=accept(sd,(struct sockaddr *)&client,&slen))) {
perror("accept() failed");
exit(1);
}
cl_add(&clients,csd,client); // Client in ne Liste schieben
FD_SET(csd,&rfds);
maxfd = MAX(maxfd,csd);
}
(...)
}
```

- `select()` reagiert, wenn auf dem Server-Socket `sd` eine Verbindung angezeigt wird

- `accept()` akzeptiert die Verbindung und erzeugt einen neuen Socket, der die Verbindung zum Client repräsentiert
- Der Server-Socket bleibt aktiv – es könnte ja noch ein Client kommen
- Der neue Socket muss natürlich in das Set der zu überwachenden File-Deskriptoren, falls uns der Client mal was schickt

Wenn uns nun ein Client was schickt, müssen wir

- den richtigen Socket raussuchen,
- die Daten vom Client empfangen und anschließend
- die empfangene Nachricht an alle anderen Clients weiterleiten.

Die Clients verwaltet man sinnvoller Weise in einer Liste (etwas ekelhaft mit C). Hat man den richtigen Socket gefunden, kann man mittels `recv()` und `send()` empfangen und senden:

```
while (1) {
(...)
len = recv(cl.sd,inbuff,BUFFLEN,0);
(...)
len = send(cl.sd,outbuff,strelen(outbuff),0);
(...)
}
```

- Prinzipiell kann man hier anstelle von `recv()` und `send()` auch die Syscalls `read()` und `write()` nutzen, da im Gegensatz zum verbindungslosen UDP Sender und Empfänger schon feststehen
- `recv()` und `send()` sind aber zu bevorzugen, da hier bestimmte Ausnahmen (z. B. Verbindung unterbrochen) sinnvoll signalisiert werden

Der Client Der Client ist viel leichter:

```
remote.sin_family      = AF_INET;
remote.sin_port        = htons(SERVERPORT);
remote.sin_addr.s_addr = htonl(SERVERIP);
if (0 > (sd=socket(AF_INET,SOCK_STREAM,IPPROTO_TCP))) {
perror("socket() failed");
exit(1);
}
if (0 > connect(sd,(struct sockaddr *)&remote,sizeof(remote))) {
perror("connect() failed");
exit(1);
}
```

- Sofern wir als Client unseren Absenderport nicht angeben wollen (sondern uns vom Betriebssystem einen zuweisen lassen wollen), können wir auf ein `bind()` verzichten.
- Wir brauchen jetzt aber in jedem Fall ein `connect()`, um uns mit dem Server zu verbinden
- `connect()` muss natürlich gesagt werden, mit wem wir uns verbinden wollen (→ `struct sockaddr_in`)

Fast geschafft, es fehlt nur noch das Senden und Empfangen von Nachrichten:

```
(...)
while(1) {
rfd = rfds;
if (0 > select(maxfd+1,&rfd,NULL,NULL,NULL)) {
perror("select() failed");
exit(1);
}
if (FD_ISSET(STDIN_FILENO,&rfd)) {
if (NULL == (s=fgets(buffer,sizeof(buffer)-1,stdin)))
continue;
if (0 >= (len=send(sd,buffer,MIN(strlen(buffer)+1,BUFFLEN),0))) { (...) }
}
if (FD_ISSET(sd,&rfd)) {
if (0 >= (len=recv(sd,buffer,BUFFLEN-1,0))) { (...) }
fprintf(stdout,">> % s\n",buffer);
}
}
```

Fast geschafft, es fehlt nur noch das Senden und Empfangen von Nachrichten:

```
(...)
while(1) {
rfd = rfds;
if (0 > select(maxfd+1,&rfd,NULL,NULL,NULL)) {
perror("select() failed");
exit(1);
}
if (FD_ISSET(STDIN_FILENO,&rfd)) {
if (NULL == (s=fgets(buffer,sizeof(buffer)-1,stdin)))
continue;
if (0 >= (len=send(sd,buffer,MIN(strlen(buffer)+1,BUFFLEN),0))) { (...) }
}
if (FD_ISSET(sd,&rfd)) {
if (0 >= (len=recv(sd,buffer,BUFFLEN-1,0))) { (...) }
fprintf(stdout,">> % s\n",buffer);
}
}
```

Jetzt wird gechattet::

- Ladet Euch den Client runter: `http://moepi.net/private/tcpchat_client.tar`
- Ein Server läuft auf `129.187.145.241` Port 6112
- Der erste, der den Server abschießt, kriegt ein Bier!

4.5. Network Address Translation (NAT)

4.5.0.1. Network Address Translation (NAT)

In Kapitel 3 haben wir gelernt, dass

- IP-Adressen zur End-zu-End-Adressierung verwendet werden,
- aus diesem Grund global eindeutig sind und
- speziell die heute hauptsächlich verwendeten IPv4-Adressen sehr knapp sind.

Frage: Müssen IP-Adressen immer eindeutig sein? In Kapitel 3 haben wir gelernt, dass

- IP-Adressen zur End-zu-End-Adressierung verwendet werden,
- aus diesem Grund global eindeutig sind und
- speziell die heute hauptsächlich verwendeten IPv4-Adressen sehr knapp sind.

Antwort: Nein, IP-Adressen müssen nicht eindeutig sein, wenn

- keine Kommunikation mit im Internet befindlichen Hosts möglich sein muss oder
- die nicht eindeutigen privaten IP-Adressen auf geeignete Weise in öffentliche Adressen übersetzt werden.

Definition: NAT:

Als Network Address Translation (NAT) bezeichnet man allgemein Techniken, welche es ermöglichen, N private (nicht global eindeutige) IP-Adressen auf M globale (weltweit eindeutige) IP-Adressen abzubilden.

- $N \leq M$: Die Übersetzung geschieht statisch oder dynamisch indem jeder privaten IP-Adresse mind. eine öffentliche IP-Adresse zugeordnet wird.
- $N > M$: In diesem Fall wird eine öffentliche IP-Adresse von mehreren Computer gleichzeitig genutzt. Eine eindeutige Unterscheidung kann mittels Port-Multiplexing erreicht werden. Der häufigste Fall ist $M = 1$, z. B. ein privater DSL-Anschluss.

Was sind private IP-Adressen? Private IP-Adressen sind spezielle Adressbereiche, welche

- zur privaten Nutzung ohne vorherige Registrierung freigeben sind,
- deswegen in unterschiedlichen Netzen vorkommen können,
- aus diesem Grund nicht eindeutig und zur End-Zu-End-Adressierung zwischen öffentlich erreichbaren Nezten geeignet sind und
- daher IP-Pakete mit privaten Empfänger-Adressen von Routern im Internet nicht weitergeleitet werden (oder werden sollten).

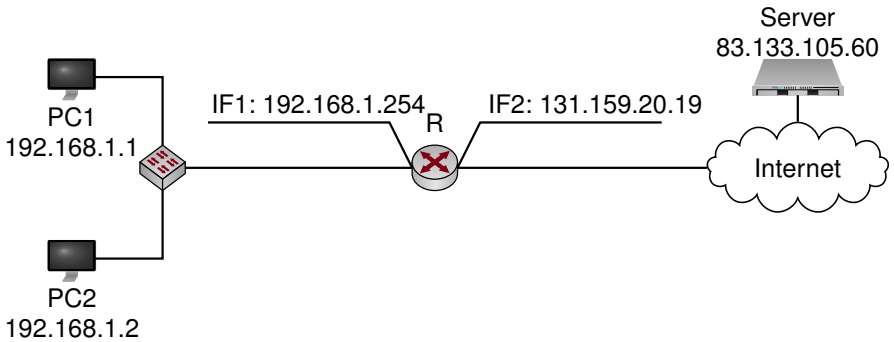
Die privaten Adressbereiche sind:

- 10.0.0.0 / 8
- 172.16.0.0 / 18
- 169.254.0.0 / 16
- 192.168.0.0 / 16

Der Bereich 169.254.0.0 / 16 wird zur automatischen Adressvergabe (Automatic Private IP Adressing) genutzt:

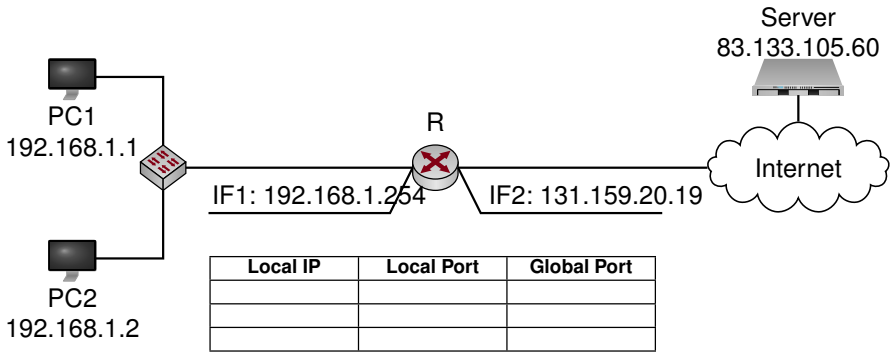
- Startet ein Computer ohne statisch vergebene Adresse, versucht dieser, einen DHCP-Server zu erreichen.
- Kann kein DHCP-Server gefunden werden, vergibt das Betriebssystem eine zufällig gewählte Adresse aus diesem Adressblock.
- Schlägt anschließend die ARP-Auflösung zu dieser Adresse fehl, wird angenommen, dass diese Adresse im lokalen Subnetz noch nicht verwendet wird. Andernfalls wird eine andere Adresse gewählt und der Vorgang wiederholt.

Wie funktioniert NAT im Detail? Im Allgemeinen übernehmen Router die Netzwerkadressübersetzung:



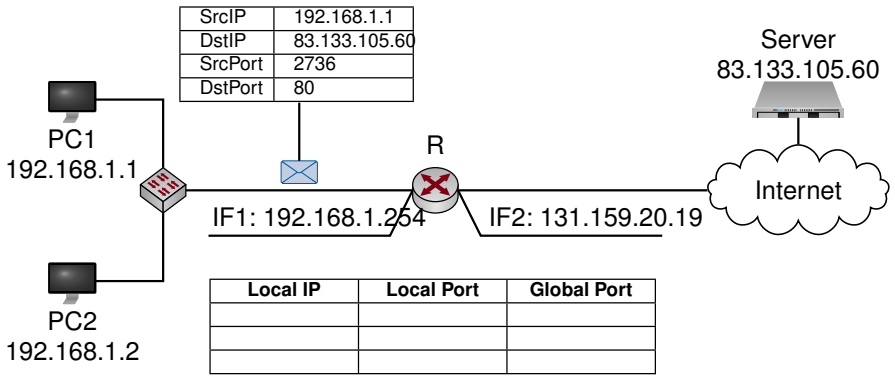
- PC1, PC2 und R können mittels privater IP-Adressen im Subnetz 192.168.1.0/24 miteinander kommunizieren.
- R ist über seine öffentliche Adresse 131.159.20.19 global erreichbar.
- PC1 und PC2 können wegen ihrer privaten Adressen nicht direkt mit anderen Hosts im Internet kommunizieren.
- Hosts im Internet können ebensowenig PC1 oder PC2 erreichen – selbst dann, wenn sie wissen, dass sich PC1 und PC2 hinter R befinden und die globale Adresse von R bekannt ist.

PC1 greift auf eine Webseite zu, welche auf dem Server mit der Adresse 83.133.105.60 liegt:



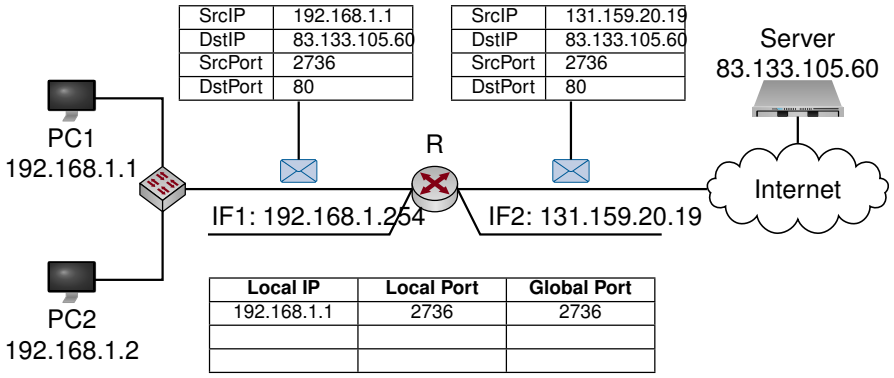
- Die NAT-Tabelle von R sei zu Beginn leer.

PC1 greift auf eine Webseite zu, welche auf dem Server mit der Adresse 83.133.105.60 liegt:



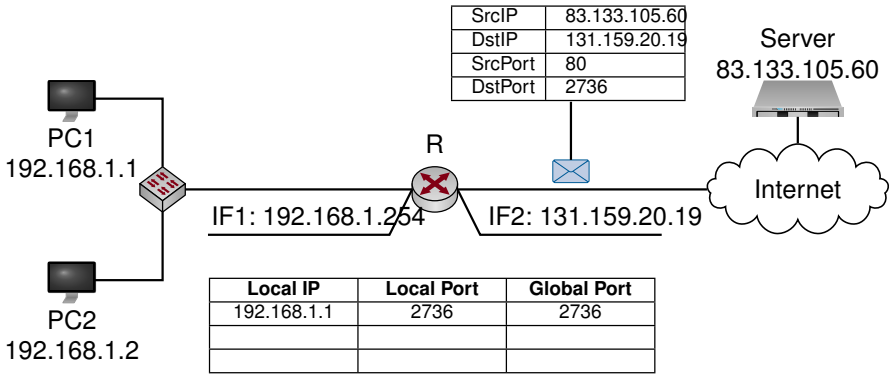
- Die NAT-Tabelle von R sei zu Beginn leer.
- PC1 sendet ein Paket (TCP SYN) an den Server:
  - PC1 verwendet seine private IP-Adresse als Absenderadresse
  - Der Quellport wird von PC1 zufällig im Bereich $[1024, 65535]$ gewählt (sog. Ephemeral Ports)
  - Der Zielport ist durch das Application Layer Protocol vorgegeben (80 = HTTP)

PC1 greift auf eine Webseite zu, welche auf dem Server mit der Adresse 83.133.105.60 liegt:



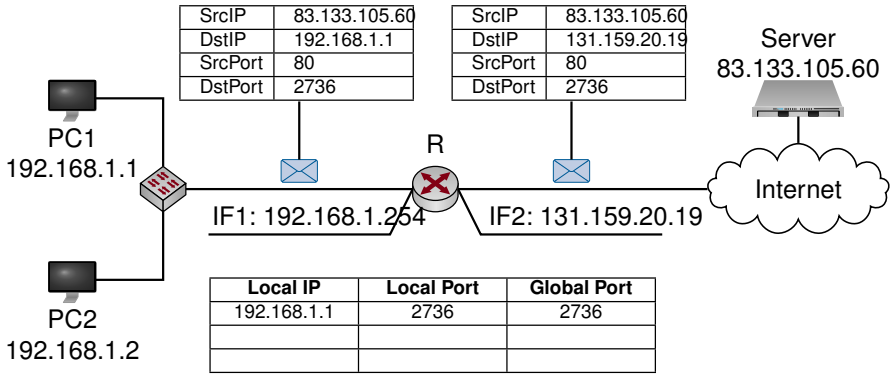
- Die NAT-Tabelle von R sei zu Beginn leer.
- PC1 sendet ein Paket (TCP SYN) an den Server:
  - PC1 verwendet seine private IP-Adresse als Absenderadresse
  - Der Quellport wird von PC1 zufällig im Bereich $[1024, 65535]$ gewählt (sog. Ephemeral Ports)
  - Der Zielport ist durch das Application Layer Protocol vorgegeben (80 = HTTP)
- Adressübersetzung an R:
  - R tauscht die Absenderadresse durch seine eigene globale Adresse aus
  - Sofern der Quellport nicht zu einer Kollision in der NAT-Tabelle führen würde, wird dieser beibehalten (andernfalls wird dieser ebenfalls ausgetauscht)
  - R erzeugt einen neuen Eintrag in seiner NAT-Tabelle, welche die Änderungen an dem Paket dokumentieren

Antwort vom Server an PC1



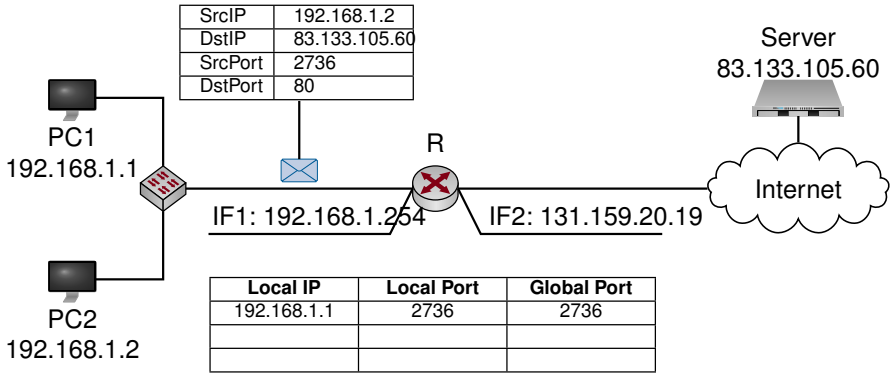
- Der Server generiert eine Antwort:
  - Der Server weiß nichts von der Adressübersetzung und hält R für PC1
  - Die Empfängeradresse ist daher die öffentliche IP von R, der Zielport der von R übersetzte Quellport aus der vorherigen Nachricht

Antwort vom Server an PC1



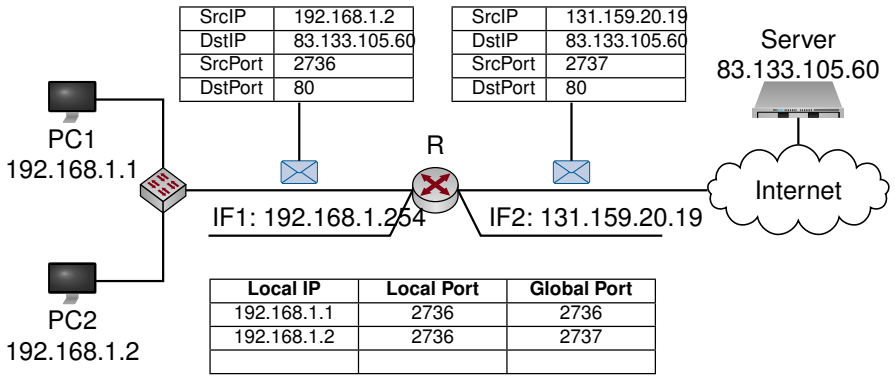
- Der Server generiert eine Antwort:
  - Der Server weiß nichts von der Adressübersetzung und hält R für PC1
  - Die Empfängeradresse ist daher die öffentliche IP von R, der Zielport der von R übersetzte Quellport aus der vorherigen Nachricht
- R mach die Adressübersetzung rückgängig
  - In der NAT-Tabelle wird nach der Zielportnummer in der Spalte Global Port gesucht, dieser in Local Port zurückübersetzt und die Ziel-IP des Pakets gegen die private IP-Adresse von PC1 ausgetauscht
  - Das so modifizierte Paket wird an PC1 weitergeleitet
  - Wie der Server weiß auch PC1 nichts von der Adressübersetzung

PC2 greift nun ebenfalls auf den Server zu:



- PC2 sendet ebenfalls ein Paket (TCP SYN) an den Server:
  - Rein zufällig wählt PC2 denselben Quell-Port wie PC1 (Portnummer 2736)

PC2 greift nun ebenfalls auf den Server zu:



- PC2 sendet ebenfalls ein Paket (TCP SYN) an den Server:
  - Rein zufällig wählt PC2 denselben Quell-Port wie PC1 (Portnummer 2736)
- Adressübersetzung an R:
  - R bemerkt, dass es bereits einen zu PC1 gehörenden Eintrag für den lokalen Port 2736 gibt
  - R erzeugt einen neuen Eintrag in der NAT-Tabelle, wobei für den globalen Port ein zufälliger Wert gewählt wird (z. B. der ursprüngliche Port von PC2 + 1)
  - Das Paket von PC2 wird entsprechend modifiziert und an den Server weitergeleitet
- Aus Sicht des Servers hat der „Computer“ R einfach zwei TCP-Verbindungen aufgebaut.

Ein Router könnte in die NAT-Tabelle zusätzliche Informationen aufnehmen:

- Ziel-IP und Ziel-Port
- Das verwendete Protokoll (TCP, UDP)
- Die eigene globale IP (sinnvoll, wenn ein Router mehr als eine globale IP besitzt)

In Abhängigkeit der gespeicherten Informationen unterscheidet man unterschiedliche Typen von NAT. Die eben diskutierte Variante (zzgl. eines Vermerks des Protokolls in der NAT-Tabelle) bezeichnet man als Full Cone NAT. **Eigenschaften von Full Cone NAT:**

- Bei eingehenden Verbindungen findet keine Prüfung der Absender-IP oder des Absender-Ports statt, da die NAT-Tabelle nur den Ziel-Port und die zugehörige IP-Adresse bzw. Portnummer im lokalen Netz enthält.
- Existiert also einmal ein Eintrag in der NAT-Tabelle, so ist ein interner Host aus dem Internet über diesen Eintrag auch für jeden erreichbar, der ein TCP- bzw. UDP-Paket an die richtige Portnummer sendet.

Andere NAT-Varianten:

- Port Restricted NAT
- Address Restricted NAT
- Port and Address Restricted NAT
- Symmetric NAT

Allgemeine Anmerkungen

- Ist NAT eine Firewall?
  - Nein.
  - Restriktive NAT-Varianten bieten zwar insofern einen grundlegenden Schutz, da sie eingehende Verbindungen ohne vorherigen Verbindungsaufbau aus dem lokalen Netz heraus erlauben, dies sollte aber nicht mit den Funktionen einer Firewall verwechselt werden.
  - Eine darüber hinausgehende Filterung von Verbindungen (wie es bei einer Firewall der Fall wäre) findet nicht statt.
- Wie viele Einträge kann eine NAT-Tabelle fassen?
  - Im einfachsten Fall (Full Cone NAT) beträgt die theoretische Maximalgrenze ca. 2^{16} pro Transportprotokoll (TCP und UDP) und pro globaler IP-Adresse.
  - Bei komplexeren NAT-Typen sind durch die Aufnahme der Ziel-Ports mehr Kombinationen möglich.

  - In der Praxis ist die Größe durch die Fähigkeiten des Routers beschränkt (einige 1000 Mappings).

- Werden Mappings aus der NAT-Tabelle wieder gelöscht?
  - Dynamisch erzeugte Mappings werden nach einer gewissen Inaktivitätszeit gelöscht.
  - U. U. entfernt ein NAT-fähiger Router auch Mappings sofort, wenn er einen TCP-Verbindungsabbau erkennt (implementierungsabhängig).

- Können Einträge in der NAT-Tabelle auch von Hand erzeugt werden
  - Ja, diesen Vorgang nennt man Port Forwarding.
  - Auf diese Weise wird es möglich, hinter einem NAT einen auf einem bestimmten Port öffentlich erreichbaren Server zu betreiben.

NAT und ICMP

- NAT verwendet Portnummern des Transportprotokolls
- Was ist, wenn das Transportprotokoll keine Portnummern hat oder IP-Pakete ohne TCP-/UDP-Header verschickt werden, z. B. ICMP?

NAT und ICMP

- NAT verwendet Portnummern des Transportprotokolls
- Was ist, wenn das Transportprotokoll keine Portnummern hat oder IP-Pakete ohne TCP-/UDP-Header verschickt werden, z. B. ICMP?

Antwort: Die ICMP-ID kann anstelle der Portnummern genutzt werden. **NAT und ICMP**

- NAT verwendet Portnummern des Transportprotokolls
- Was ist, wenn das Transportprotokoll keine Portnummern hat oder IP-Pakete ohne TCP-/UDP-Header verschickt werden, z. B. ICMP?

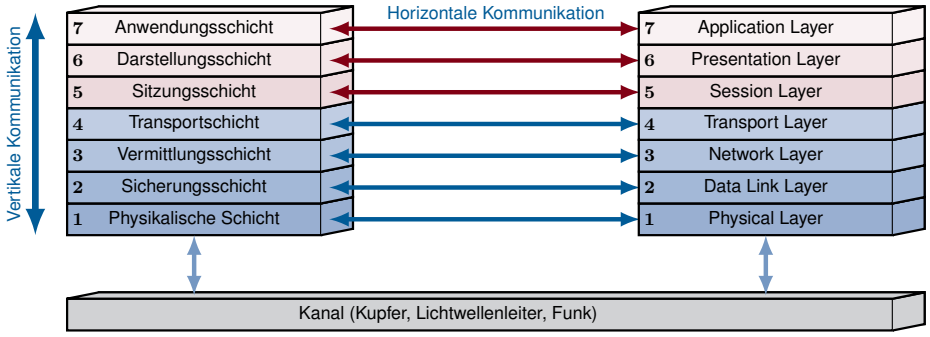
Antwort: Die ICMP-ID kann anstelle der Portnummern genutzt werden. **Problem:** Traceroute funktioniert mit manchen Virtualisierungslösungen nicht, z. B. wenn ältere Versionen der Virtualbox-NAT-Implementierung verwendet werden.

- Traceroute basiert auf ICMP-TTL-Exceeded-Nachrichten
- Diese Nachrichten haben (anders als ein ECMP Echo Reply) keine ICMP-ID.
- Das liegt daran, dass jedes beliebige IP-Paket (nicht zwangsläufig ein ICMP-Echo-Request) ein ICMP-TTL-Exceeded auslösen kann und dieses (wie im Fall eines TCP-Pakets) natürlich keine ICMP-ID besitzt.
- Stattdessen trägt der Time-Exceeded den vollständigen IP-Header und die ersten 8 Byte der Payload des Pakets, welches den Time-Exceeded ausgelöst hat.
- Eine NAT-Implementierung müsste nun (genau wie Sie in Assignment 2) im Fall eines TTL-Exceeded in diesen ersten 8 Byte nach der ICMP-ID eines Echo-Requests oder aber nach den Portnummern eines Transportprotokolls suchen, um die Übersetzungs rückgängig machen zu können.
- Genau diese Rückübersetzung führen ältere Versionen der NAT-Implemnentierung von Virtualbox nicht durch.

5. Sitzungs-, Darstellungs- und Anwendungsschicht

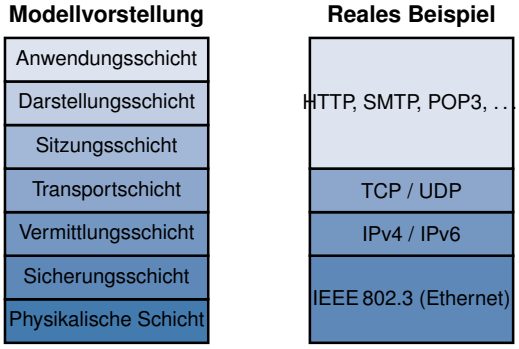
5.1. Motivation

Einordnung im ISO/OSI-Modell



Modell und Realität

- Dienste der Sitzungs- und der Darstellungsschicht sind in einzelnen Fällen in Form standardisierter Protokolle implementiert.
- In anderen Fällen sind Funktionen, die der Sitzungs- bzw. der Darstellungsschicht zuzuordnen sind, in die Anwendung integriert.



- Die Standards der ITU-Serie X.200 beschreiben Dienste der sieben OSI-Schichten, sowie Protokolle zur Erbringung dieser Dienste.
- Die in diesen Standards vorgenommene Strukturierung ist nützlich.
- Etliche OSI-Protokolle haben in der Praxis kaum Bedeutung.

Im Folgenden werden wir

- die Aufgaben der Sitzungs- und Darstellungsschicht erläutern,
- beispielhaft einige Protokolle kennenlernen, welche den Schichten 5 und 6 zugeordnet werden können,
- sowie wichtige Protokolle der Anwendungsschicht erläutern.

5.2. Sitzungsschicht

5.2.1. Betriebsmodi

Die Sitzungsschicht kann nach X.200 in zwei verschiedenen **Modi** betrieben werden:

- **Connection-Oriented Mode:**
  Die Sitzungsschicht baut eine Verbindung zwischen den Kommunikationspartnern auf, die über die Dauer einzelner Datentransfers hinweg erhalten bleibt.
  Es werden die Phasen Verbindungsaufbau, Datentransfer und Verbindungsabbau unterschieden.
  Die Sitzungsschicht kann in diesem Fall vielfältige Aufgaben erfüllen.
  Die unterstützte Funktionalität beinhaltet die Koordination mehrerer beteiligter Parteien bezüglich Datenfluss, verwendeter Dienste, Aushandlung diverser Parameter für den Kommunikationsablauf und Identifikation der Verbindungen.
- **Connectionless Mode:**
  In diesem Fall wird durch die Sitzungsschicht nur ein sehr einfacher Dienst erbracht.
  Von der dienstnehmenden Schicht werden die zu übertragenden Daten entgegengenommen, zusammen mit Informationen zur Adressierung und zu den Dienstgüteanforderungen, und an die Transportschicht weitergereicht.

Hinweis:

Eine Verbindung der Sitzungsschicht ist nicht gleichbedeutend mit einer Verbindung der Transportschicht! Eine Session kann beispielsweise nacheinander mehrere TCP-Verbindungen beinhalten.

5.2.2. Dienste der Sitzungsschicht

Definition (Session):

Eine Session beschreibt die Kommunikation zwischen zwei oder mehreren Teilnehmern, mit definiertem Anfang und Ende und sich daraus ergebender Dauer.

Um für die dienstnehmende Schicht (Darstellungsschicht) eine Dialogführung zu ermöglichen, müssen gegebenenfalls mehrere Transportschicht-Verbindungen verwendet und kontrolliert werden. Dies kann auch die Behandlung abgebrochener und wiederaufgenommer TCP-Verbindungen beinhalten.
Im Connection-Oriented Mode werden verschiedene Dienste angeboten:

- **Aufbau** und **Abbau** von Sessions,

- normaler und beschleunigter **Datentransfer** [1],
- Token-Management zur **Koordination** der Teilnehmer,
- **Synchronisation** und Resynchronisation,
- **Fehlermeldungen** und Aktivitätsmanagement, sowie
- **Erhaltung** und **Wiederaufnahme** von Sessions nach Verbindungsabbrüchen.

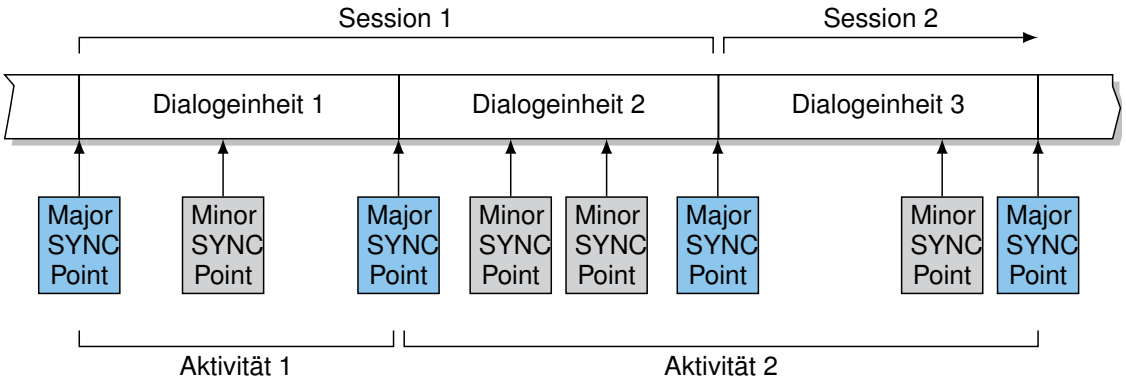
Die Sitzungsschicht stellt im Connectionless Mode folgenden einfachen Dienst bereit:

- **Datentransfer**

5.2.3. Funktionseinheiten der Sitzungsschicht

Die Dienste der Sitzungsschicht werden gemäß dem Standard X.215 in Funktionseinheiten gruppiert:

Bezeichnung	Beschreibung
Kernel	Bereitstellung von Basisfunktionen
Halb-duplex	Abwechselndes Senden und Empfangen
Duplex	Gleichzeitiges Senden und Empfangen
Negotiated Release	Beenden der Session mit gegenseitigem Bestätigen
Expedit Data	Beschleunigter Datentransfer
Activity Management	Logische Strukturierung der Session
Major Synchronization	Interne Strukturierung der Sessions in Dialogeinheiten
Minor Synchronization	Interne Strukturierung der Dialogeinheiten



Kombination von Funktionseinheiten

Unterschied zwischen Aktivitätsmanagement und Synchronisation:

Aktivitäten bestehen aus mehreren Dialogeinheiten und können jederzeit unterbrochen und später (in der gleichen oder auch einer anderen Sitzung) wieder aufgenommen werden; d.h. eine Aktivität kann auch über mehrere Sessions hinweg existieren.
Synchronisation erfolgt durch sog. Synchronisationspunkte, an welchen Sessions oder Aktivitäten wieder aufgesetzt werden können, oder an welchen eine Resynchronisation (Zurücksetzen)

[1] Expedited Data Transfer: dringliche Daten, wie Alarme oder Interrupts

möglich ist. Die Kommunikationspartner können hier prüfen, wie weit die Kommunikation, z. B. eine Datenübertragung, vorangeschritten ist.

Folgende Kombinationen von Funktionseinheiten sind vorgesehen:

- BCS Basic Combined Subset
  Kernel, Halb-Duplex/Duplex
- BSS Basic Synchronized Subset
  Kernel, Halb-Duplex, Negotiated Release, Minor/Major synchronize, Resynchronize
- BAS Basic Activity Subset
  Kernel, Halb-Duplex, Minor sychronize, Execptions, Activity Management

Eine gültige Kombination wird vor Beginn der Session zwischen den Kommunikationspartnern ausgehandelt.

5.2.4. Synchronisation

Es werden folgende Arten von Synchronisationspunkten unterschieden:

- Major Synchronisationspunkte
- Minor Synchronisationspunkte

Definition (Major und Minor Synchronisationspunkte):

Major Synchronisationspunkte werden verwendet, um die Struktur der ausgetauschten Daten in eine Serie von Dialogeinheiten zu zerlegen. Sie müssen explizit bestätigt werden.
Minor Synchronisationspunkte werden für die Strukturierung innerhalb dieser Dialogeinheiten verwendet. Sie können bestätigt werden.
Bis zu einem solchen Punkt versandte Daten werden nicht von einem Resynchronisationsprozess verworfen.

Um die Kommunikation zu steuern, werden Marken (Token) verwendet:

- Datenmarken (“data token” - geben an, wer bei der Verwendung des Halb-duplex-Betriebs senden darf)
- Beendigungsmarken (“release token” - beenden eine Sitzung)
- Synchronisationsmarken (“synchronize-minor token”)
- Aktivitätsmarken (“activity-major token”)

5.2.5. QoS und Performance

Quality of Service im Session Layer Auf der Sitzungsschicht kann zwischen verschiedenen QoS-Parametern unterschieden werden:

- Service Parameter
- Performance Parameter

Es werden zunächst die verschiedenen Service Parameter erläutert.

Definition (Service Parameter):

Protection: Beschreibt den Schutz gegen unautorisiertes Lesen oder Manipulation von Nutzerinformationen
Priorität: Beschreibt die Relation zwischen den einzelnen Sessions (Aufteilung von Ressourcen)
Resilience: Beschreibt die Wahrscheinlichkeit, dass eine Session ordnungsgemäß durchgeführt werden kann bzw. wie viele Fehler tolieriert werden können

Performance Parameter des Session Layers

- Aufbau der Session: Establishment Delay und Establishment Failure Probability
- Datentransfer
  - Durchsatz und Transit Delay
  - Residual Error Rate und Transfer Failure Probability
- Abbau der Session: Release Delay und Release Failure Probability

Definition (Residual Error Rate):

Die RER bezeichnet das Verhältnis der Menge fehlerhaft übertragener, verlorener oder dupliziert empfangener Daten zur Gesamtmenge der gesendeten Daten:

$$\text{RER} = \frac{S_e + S_l + S_x}{S}.$$

- S_e beschreibt fehlerhaft übertragene Daten, S_l verloren gegangene Daten und S_x dupliziert empfangenen Daten.
- S beschreibt die Gesamtmenge gesendeter Daten.
- Die Gesamtmenge S gesendeter Daten setzt sich zusammen aus $S_s + S_e + S_l + S_x$, mit der Menge korrekt übertragener Daten S_s.
- Die Menge empfangener Daten setzt sich zusammen aus $S_s + S_e + S_x$.

5.3. Darstellungsschicht

5.3.1. Aufgaben der Darstellungsschicht

Die Aufgabe der Darstellungsschicht (engl. Presentation Layer) ist es, den Kommunikationspartnern eine einheitliche Interpretation der Daten zu ermöglichen, d. h. Daten in einem einheitlichen Format zu übertragen. Der Darstellungsschicht sind grundsätzlich folgende Aufgaben zugeordnet:

- die Darstellung der Daten (Syntax),
- die Datenstrukturen zur Übertragung der Daten
- die Darstellung der Aktionen an diesen Datenstrukturen, sowie
- Datentransformationen.

Hinweis:

Die Darstellung auf Schicht 6 muss nicht der Darstellung auf Schicht 7 (Anwendungsschicht) entsprechen. Die Darstellungsschicht ist für die Syntax der Nutzdaten verantwortlich, die Semantik verbleibt bei den Anwendungen.
Unter Syntax versteht man die Darstellung von Daten nach bestimmten Regeln (Grammatik). Werden Daten durch Bedeutung ergänzt, spricht man von Informationen, dies ist die Aufgabe der Semantik.

- Anwendungen sollen syntaxunabhängig miteinander kommunizieren können.
- Anwendungsspezifische Syntax kann von der Darstellungsschicht in eine einheitliche Form umgewandelt und dann übertragen werden.

Den grundsätzlichen Aufgaben der Darstellungsschicht lassen sich konkrete Funktionen zuordnen:

- Quellenkodierung und Datenkompression
- Umkodierungen und Übersetzung zwischen Datenformaten
- Strukturierte Darstellung von Informationen
- Ver- und Entschlüsselung

Existierende Protokolle lassen sich nicht immer eindeutig einer Schicht zuordnen. Relevante Protokolle (wie beispielsweise TLS) beinhalten sowohl Funktionen, die üblicherweise der Schicht 5 zugeordnet werden, als auch Funktionen, die üblicherweise der Schicht 6 zugeordnet werden. **Frage:** Warum lässt sich das TLS-Protokoll sowohl der Schicht 5 als auch der Schicht 6 zuordnen?

- Der TLS Handshake dient dem Aushandeln von Sitzungsschlüssel (engl. Session Keys).
- Dies entspricht dem Aushandeln der Kommunikationsparameter einer Sitzung und kann somit der Sitzungsschicht zugeordnet werden.
- Bei den Funktionen Kompression sowie Ver- und Entschlüsseln von Nutzdaten handelt es sich um Funktionen, die der Darstellungsschicht zugeordnet werden.
- TLS lässt sich somit sowohl der Schicht 5 als auch der Schicht 6 zuordnen.

5.3.2. Datenkompression und Umkodierung

Datenkompression und Umkodierung gehört zu den wichtigsten Funktionen der Darstellungsschicht. Es ist wichtig, die zugrundeliegenden Prozessen zu unterscheiden.

Definition (Umkodierung):

Umkodierung oder Transkodierung von Daten beschreibt den Prozess der Überführung von einer Darstellung in eine andere. Hierbei sind keine Einschränkungen wie bezüglich der Anzahl verwendeter Zeichen o. ä. nötig. Eine gültige Umkodierung kann gegebenen Nutzdaten in ihrem Umfang auch erweitern.

Definition (Kompression):

Unter Datenkompression versteht man die Entfernung von Redundanz. Die komprimierte Nachricht ist i. A. kürzer als das Original.

Redundanzfreie Informationen besitzen auch nach der Kompression dieselbe Größe. Da bei einer Kompression üblicherweise zusätzliche Informationen (Header) hinzugefügt werden müssen, kann zum Beispiel bei sehr kurzen Informationen vorkommen, dass die komprimierte Version länger ist.

- Wir wollen im Folgenden näher auf Kompression eingehen.
- Als Beispiel für ein Kompressionsverfahren werden wir den Huffman-Algorithmus betrachten.
- Zudem wollen wir wichtige Eigenschaften des Huffman-Algorithmus näher analysieren.

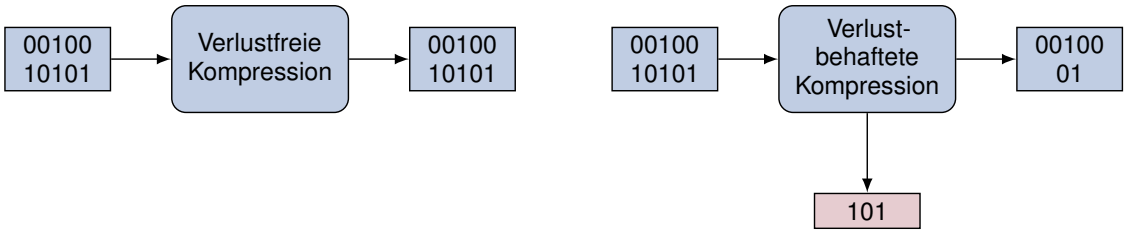
Einteilung von Datenkompression und Umkodierung Kodierungsverfahren können unterschieden werden nach

- Fixed-Length Code bei welchen alle Zeichen mit der gleichen Anzahl Bits kodiert werden, z.B. ASCII (8 Bits) oder UNICODE (16 Bits).
- Variable-Length Code bei welchen alle Zeichen abhängig von ihrer Auftrittswahrscheinlichkeit mit einer Kodierungslänge korreliert werden.

Es lassen sich zwei grundlegende Arten der Kompression unterscheiden:

- Verlustfreie Komprimierung (engl. Lossless Data Compression): Die dekodierten Daten können identisch zum Original wieder hergestellt werden, ohne dass Informationen verloren gehen oder verändert werden (wie z.B. beim ZIP-Dateiformat, das den Einsatz unterschiedlicher verlustfreier Kompressionsalgorithmen ermöglicht).
- Verlustbehaftete Komprimierung (engl. Lossy Data Compression): Bei der Dekodierung gehen Informationen der ursprünglichen Daten verloren und können nicht mehr vollständig richtig dekodiert werden (siehe z.B. das verlustbehaftete Kompressionsverfahren JPEG).

Insbesondere bei vom Menschen analog wahrgenommenen Daten (Audio, Bilder, Video) lassen sich Informationen verwerfen, die ohnehin nicht wahrnehmbar sind (z. B. Frequenzen über 20000 Hz)



Beispiel: Kompression mittels Huffman-Codes

- Viele Protokolle komprimieren Daten vor dem Senden (Quellenkodierung).
- TLS beispielsweise bietet optional Kompressionsmethoden. Diese werden vor der Verschlüsselung angewandt. (Warum davor?)
- Ein häufig (u. a. von TLS) verwendetes Kompressionsverfahren für Texte ist der Huffman-Code.

Grundlegende Idee der Huffman-Kodierung:

- Nicht alle Textzeichen treten mit derselben Häufigkeit auf, z. B. tritt der Buchstabe „E“ in der deutschen Sprache mit einer Häufigkeit von 17.4 % gefolgt von „N“ mit 9.78 % auf.
- Anstelle Zeichen mit uniformer Codewortlänge zu kodieren, (z. B. ASCII-Code), weise häufigen Zeichen kürzere Codewörter zu.

Bei der Huffman-Kodierung handelt es sich also um ein

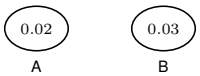
- verlustfreies Kompressionsverfahren.

Konstruktion eines Huffman-Codes: Gegeben Sei das Alphabet $\mathcal{A} = \{A, B, C, D, E, F, G, H\}$ sowie Auftrittswahrscheinlichkeiten $\Pr[X = z]$ für alle Zeichen $z \in \mathcal{A}$. Es wird außerdem vorausgesetzt, dass die einzelnen Zeichen unabhängig voneinander auftreten.

z	$\Pr[X = z]$
A	0.02
B	0.03
C	0.05
D	0.08
E	0.12
F	0.15
G	0.25
H	0.30

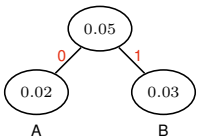
Konstruktion eines Huffman-Codes: Gegeben Sei das Alphabet $\mathcal{A} = \{A, B, C, D, E, F, G, H\}$ sowie Auftrittswahrscheinlichkeiten $\Pr[X = z]$ für alle Zeichen $z \in \mathcal{A}$. Es wird außerdem vorausgesetzt, dass die einzelnen Zeichen unabhängig voneinander auftreten.

z	$\Pr[X = z]$
A	0.02
B	0.03
C	0.05
D	0.08
E	0.12
F	0.15
G	0.25
H	0.30



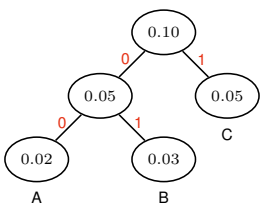
Konstruktion eines Huffman-Codes: Gegeben Sei das Alphabet $\mathcal{A} = \{A, B, C, D, E, F, G, H\}$ sowie Auftrittswahrscheinlichkeiten $\Pr[X = z]$ für alle Zeichen $z \in \mathcal{A}$. Es wird außerdem vorausgesetzt, dass die einzelnen Zeichen unabhängig voneinander auftreten.

z	$\Pr[X = z]$
A	0.02
B	0.03
C	0.05
D	0.08
E	0.12
F	0.15
G	0.25
H	0.30



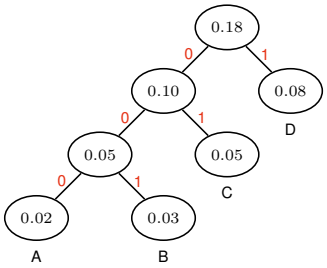
Konstruktion eines Huffman-Codes: Gegeben Sei das Alphabet $\mathcal{A} = \{A, B, C, D, E, F, G, H\}$ sowie Auftrittswahrscheinlichkeiten $\Pr[X = z]$ für alle Zeichen $z \in \mathcal{A}$. Es wird außerdem vorausgesetzt, dass die einzelnen Zeichen unabhängig voneinander auftreten.

z	$\Pr[X = z]$
A	0.02
B	0.03
C	0.05
D	0.08
E	0.12
F	0.15
G	0.25
H	0.30



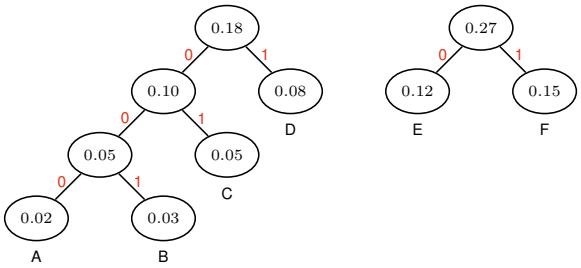
Konstruktion eines Huffman-Codes: Gegeben Sei das Alphabet $\mathcal{A} = \{A, B, C, D, E, F, G, H\}$ sowie Auftrittswahrscheinlichkeiten $\Pr[X = z]$ für alle Zeichen $z \in \mathcal{A}$. Es wird außerdem vorausgesetzt, dass die einzelnen Zeichen unabhängig voneinander auftreten.

z	$\Pr[X = z]$
A	0.02
B	0.03
C	0.05
D	0.08
E	0.12
F	0.15
G	0.25
H	0.30



Konstruktion eines Huffman-Codes: Gegeben Sei das Alphabet $\mathcal{A} = \{A, B, C, D, E, F, G, H\}$ sowie Auftrittswahrscheinlichkeiten $\Pr[X = z]$ für alle Zeichen $z \in \mathcal{A}$. Es wird außerdem vorausgesetzt, dass die einzelnen Zeichen unabhängig voneinander auftreten.

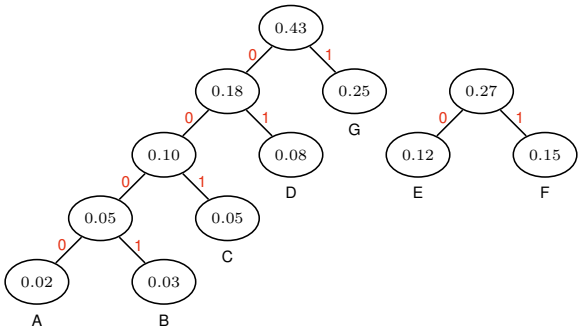
z	$\Pr[X = z]$
A	0.02
B	0.03
C	0.05
D	0.08
E	0.12
F	0.15
G	0.25
H	0.30



Konstruktion eines Huffman-Codes: Gegeben Sei das Alphabet $\mathcal{A} = \{A, B, C, D, E, F, G, H\}$ sowie Auftrittswahrscheinlichkeiten $\Pr[X = z]$ für alle Zeichen $z \in \mathcal{A}$. Es wird außerdem vor-

ausgesetzt, dass die einzelnen Zeichen unabhängig voneinander auftreten.

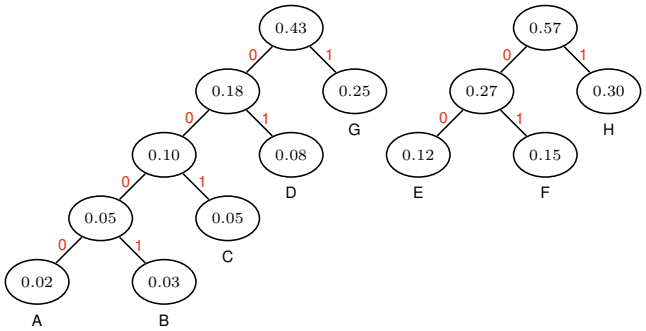
z	$\Pr[X = z]$
A	0.02
B	0.03
C	0.05
D	0.08
E	0.12
F	0.15
G	0.25
H	0.30



Konstruktion eines Huffman-Codes: Gegeben Sei das Alphabet $\mathcal{A} = \{A, B, C, D, E, F, G, H\}$ sowie Auftrittswahrscheinlichkeiten $\Pr[X = z]$ für alle Zeichen $z \in \mathcal{A}$. Es wird außerdem vor-

ausgesetzt, dass die einzelnen Zeichen unabhängig voneinander auftreten.

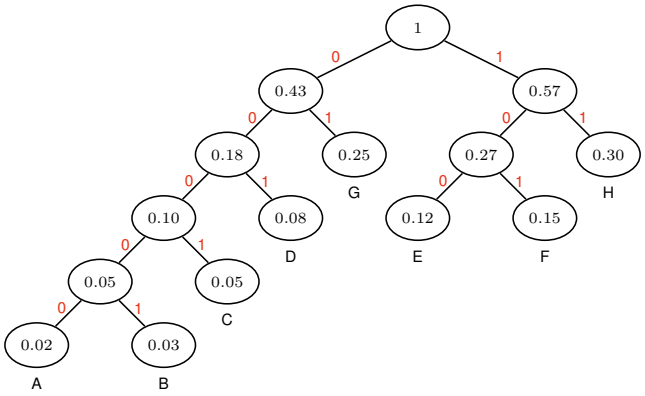
z	$\Pr[X = z]$
A	0.02
B	0.03
C	0.05
D	0.08
E	0.12
F	0.15
G	0.25
H	0.30



Konstruktion eines Huffman-Codes: Gegeben Sei das Alphabet $\mathcal{A} = \{A, B, C, D, E, F, G, H\}$ sowie Auftrittswahrscheinlichkeiten $\Pr[X = z]$ für alle Zeichen $z \in \mathcal{A}$. Es wird außerdem vor-

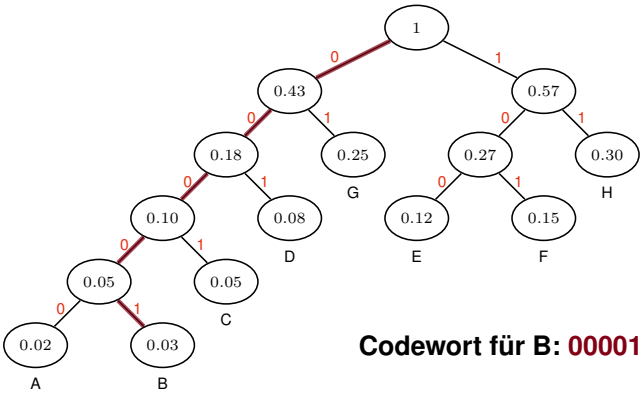
ausgesetzt, dass die einzelnen Zeichen unabhängig voneinander auftreten.

z	$\Pr[X = z]$
A	0.02
B	0.03
C	0.05
D	0.08
E	0.12
F	0.15
G	0.25
H	0.30



Konstruktion eines Huffman-Codes: Gegeben Sei das Alphabet $\mathcal{A} = \{A, B, C, D, E, F, G, H\}$ sowie Auftrittswahrscheinlichkeiten $\Pr[X = z]$ für alle Zeichen $z \in \mathcal{A}$. Es wird außerdem vorausgesetzt, dass die einzelnen Zeichen unabhängig voneinander auftreten.

z	$\Pr[X = z]$
A	0.02
B	0.03
C	0.05
D	0.08
E	0.12
F	0.15
G	0.25
H	0.30



Durchschnittliche Codewortlänge

z	$\Pr[X = z]$	Huffman-Code	Länge $l_H(z)$	„Einfacher“ Code
A	0.02	00000	5	000
B	0.03	00001	5	001
C	0.05	0001	4	010
D	0.08	001	3	011
E	0.12	010	3	100
F	0.15	011	3	101
G	0.25	10	2	110
H	0.30	11	2	111

- Uniformer Code:

$$\mathrm{E}[l(z)] = 3.0, \quad \text{da alle Codewörter gleich lang sind}$$

- Huffman-Code:

$$\mathrm{E}[l_H(z)] = \sum_{z \in \mathcal{A}} \Pr[X = z] \cdot l_H(z) = 2.6$$

$$\Rightarrow \text{Die Einsparung beträgt } 1 - \mathrm{E}[l_H(z)]/\mathrm{E}[l(z)] \approx 13\,\%$$

Anmerkungen zum Huffman-Code:

- Statische Huffman-Codes sind darauf angewiesen, dass die Auftrittswahrscheinlichkeit der Zeichen den Erwartungen entspricht.
- Zeichenhäufigkeiten können dynamisch bestimmt werden, allerdings muss dem Empfänger dann das verwendete Codebuch mitgeteilt werden.
- Längere Codewörter (z. B. ganze Wörter statt einzelner Zeichen) werden infolge der Komplexität zum Bestimmen des Codebuchs ein Problem.
- Der Huffman-Code ist ein optimaler und präfixfreier Code.

Definition (Optimaler Präfixcode):

Bei einem Präfixfreien Code sind gültige Codewörter niemals Präfix eines anderen Codeworts desselben Codes. Ein optimaler Präfixfreier Code minimiert darüber hinaus die mittlere Codewortlänge

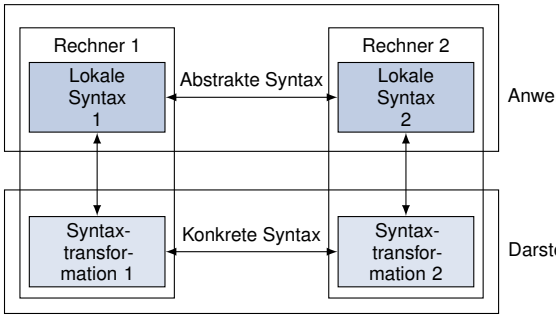
$$\sum_{i \in \mathcal{A}} p(i) \cdot |c(i)|,$$

wobei $p(i)$ die Auftrittswahrscheinlichkeit von $i \in \mathcal{A}$ und $c(i)$ die Abbildung auf ein entsprechendes Codewort bezeichnen.

- Es handelt sich zudem um einen Variable-Length Code.
- Die Huffman-Codierung zählt zu den sogenannten Entropy-Encoding Verfahren, welche im Anschluss näher analysiert werden sollen.

5.3.3. Syntax und Encoding

Einheitliche Syntax Der ITU Standard X.208 schlägt die folgende Methodik vor, um Daten mit unterschiedlichen syntaktischen Gegebenheiten zwischen zwei Systemen zu übertragen.



- Abstrakte Syntax beschreibt die Menge aller Darstellungstypen, die durch die Anwendungsprozesse definiert wurden.
- Die Kodierung dieser Darstellungstypen im lokalen System wird als lokale Syntax bezeichnet.
- Die abstrakte Syntax wird mittels einer sog. Transfersyntax zwischen den Instanzen über die Darstellungsschicht transferiert.
- Die Abbildung von einer abstrakten Syntax in eine konkrete Transfersyntax wird mittels Kodierregeln (engl. Encoding Rules) erreicht.
- Der Presentation Context bestimmt die Übergangsregeln für die De- und Enkodierung zwischen abstrakter Syntax und Transfersyntax.

Abstrakte Syntaxnotation Nummer 1: ASN.1

- ASN.1 ist eine abstrakte Syntax.
- Sie ist in der Backus-Naur-Notation gegeben und wird als Semantiksprache verwendet.

Definition (Backus-Naur-Form):

Die Backus-Naur-Form stellt eine Möglichkeit dar, die Syntax einer formalen Sprache zu beschreiben. In den BNF-Regeln sind folgende Elemente zugelassen:

- Syntaktische Variablen bzw. Nichtterminalsymbole (engl. nonterminals) dienen als Platzhalter für syntaktische Elemente. Nichtterminalsymbole werden mit <> beschrieben.
- Das Symbol **::=** dient einer Zuweisung.
- Terminalsymbole werden einzelnen Zeichen oder Zeichenketten der Wörter der Sprache bezeichnet.
- Operatoren ermöglichen die Verknüpfung von Terminalen und/oder Nichtterminalen.

Für die Verknüpfungen stehen folgenden Möglichkeiten zur Verfügung:

- Verkettung / Konkatenation
- Klammerung mehrere Ausdrücke: ( )
- Auswahl verschiedener Ausdrücke: |
- Wiederholungen einzelner Zeichen: *, um mehrere Zeichen zu wiederholen werden geschweifte Klammern { } verwendet
- optionale Ausdrücke: []

ASN.1 Beispiel

- Bei ASN.1 werden zusammengehörende Definitionen in Modulen gegliedert.
- Module können Definitionen exportieren (declarations) oder importieren (linkage)

```
ModulName DEFINITIONS ::= BEGIN
linkage
declarations
END
```

Definierbare Objekte in ASN.1 umfassen folgende Elemente:

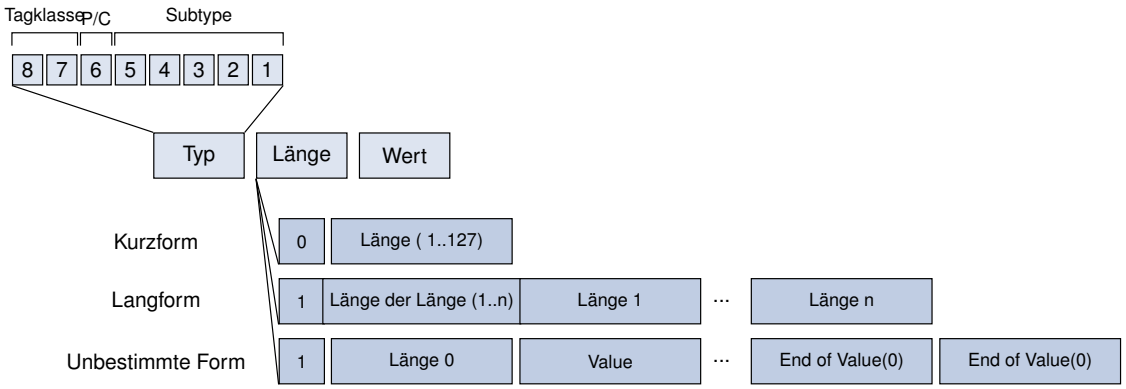
- Typen (Types), um neue Datenstrukturen zu definieren, und werden mit Großbuchstaben am Beginn gekennzeichnet (`DatenTyp`)
- Werte (Values), um die Ausprägung der Typen zu definieren, werden lediglich in Kleinbuchstaben geschrieben (`zustand`)
- Makros (Macros), um die Grammatik der Sprache ändern zu können, werden wie alle anderen Schlüsselwörter in ASN.1 mit Großbuchstaben versehen (`OBJEKT`).
- Ein Typ wir durch `TypName ::= WERT` eingeführt, Variablen werden durch `VariablenName TypName ::` beschrieben.
- ASN.1 stellt eine Vielzahl von Datentypen bereit, die hier nicht näher beschrieben werden sollen.
- Eine genaue Beschreibung der Sprache findet sich in X.208 bzw. IS 8824 (für interessierte Studenten).

Basic Encoding Rules (BER)

Definition (Basic Encoding Rules, BER):

Die Basic Encoding Rules (BER) legen eine konkrete Transfersyntax fest, die definiert, wie die Datentypen der ASN.1 bei der Übertragung dargestellt werden. Sie sind in ISO 8825 definiert. Die Reihenfolge der Bits in einem Oktett und die Ordnung der Bytes selbst beginnt mit den Most Significant Bit (MSB) am Anfang.

In BER werden die Datentypen nach ASN.1 wie folgt kodiert und werden kurz mit TLV beschrieben:



Anmerkungen zur Kodierung in BER: Typ- oder Tag-Feld Das Typ- bzw. Tag Feld enthält Informationen über die Klasse in den Bits 8 und 7, welches als class-Element bezeichnet wird. Mögliche Kodierungen sind

- Universal (**00**): ist reserviert für die Typen im ASN.1 Standard
- Application (**01**): ist gültig für eine Anwendung
- Context-Specific (**10**): ist gültig für eigene Spezifikationen
- Private (**11**): unterliegt keinen Einschränkungen, die Bedeutung geht aus dem Kontext hervor

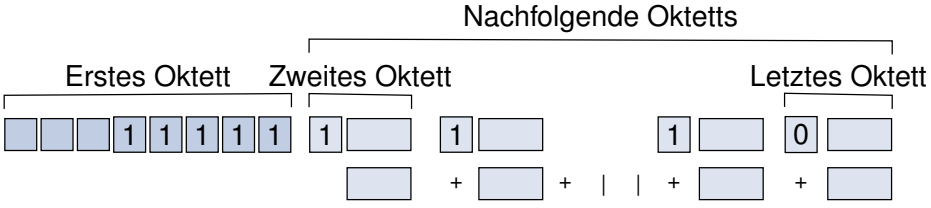
Bit Nummer 6 wird als P/C-Bit bezeichnet und gibt an, ob es sich um einen primitiven oder konstrurierten Typ handelt.

- **P/C = 1**: Der Value-Teil enthält wiederum eine BER-Struktur (TLV).
- **P/C = 0**: Der Value-Teil enthält nur ein einfaches Element

In den Bits 5 bis 1 wir der Subtype des Elements kodiert. Für die Klasse UNIVERSAL kann beispielsweise folgendes verwendet werden:

- **0**: End of Content (EOC)
- **1**: Boolean
- **2**: Integer
- Werte bis einschließlich 30 identifizieren weitere Subtypen, z.B. 4: OCTET STRING, 5: NULL
- **31**: Continued Number type: Die Identifikation des Subtypes befindet sich in einem späteren Feld

Anmerkungen zur Kodierung in BER: Continued Number Type Der Continued Number Type hat folgende Struktur:



- Bit 8 jedes Oktetts wird auf 1 gesetzt um anzuzeigen, dass weitere Subtypefelder-Felder folgen.
- Wird das 8. Bit des Oktetts auf 0 gesetzt, ist das Ende des Subtypes erreicht.
- Diese Darstellung wird verwendet, wenn der Subtype durch einen größeren Wert als 30 identifiziert wird.

Anmerkungen zur Kodierung in BER: Längenfeld Das Längenfeld gibt an, wie viele Datenoktetts folgen. Es können verschiedene Formen unterschieden werden:

- Kurzform: Das MSB (Most Significant Bit, d.h. Bit 8) wird auf 0 gesetzt. Die Bits 7 bis 1 codieren einen Wert zwischen 0 und 127, der die Länge des "ValueFeldes angibt..
- Langform: Das MSB (Most Significant Bit) wird auf 1 gesetzt und die Bits 7 bis 1 kodieren einen Wert zwischen 1 und 127, der die Anzahl nachfolgender Oktette des Längen-Felds angibt (Der Wert 127 ist reserviert für eventuelle Erweiterungen).
- Unbestimmte Form: Das MSB (Most Significant Bit) wird auf 1 gesetzt und die Bits 7 bis 1 auf 0 gesetzt. Es werden im "ValueFeld solange Oktetts übertragen bis der Delimiter (zwei auf Null gesetzte Oktette) verschickt wird. Es ist darauf zu achten, dass diese Struktur nicht in den zu versendenden Daten erscheint (s. Character Stuffing, Kapitel 4).

Frage: Wie viele Byte können mit Hilfe der jeweiligen Längenfelder übertragen werden?

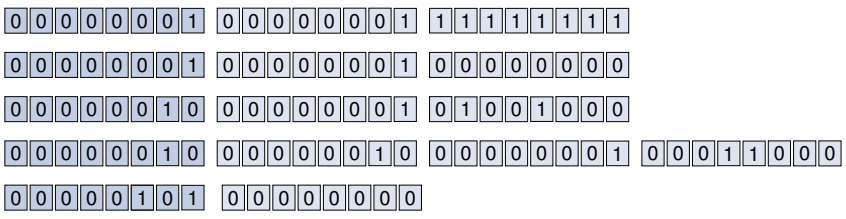
- Mit Hilfe der Kurzform: maximal 127 Byte. Die Langform erlaubt $126 * 8 = 1008$ Bit zur Kodierung der Länge: $2,47 * 10^{303}$ Byte

Frage: Welchen Vorteil bringt eine unbestimmte Form, wenn die Anzahl der zu übertragenen Bytes der Langform ausreichen würde?

- Die genaue Länge muss zu Beginn des Sendevorgangs nicht bekannt sein und die Übertragung kann beginnen, bevor die genaue Anzahl der Elemente bekannt ist.

BER: Beispiele Es folgen einige Beispiele um die Basic Encoding Rules zu verdeutlichen:

- Boolean: TRUE und FALSE
- Integer: 72 und 280
- NULL



- Eine weitere konkrete Transfersyntax stellen die Packed Encoding Rules (PER) dar.
- Sie stellen die Nachfolgeregelung der BER dar und sind wesentlich effizienter vor allem in Bezug auf die Anzahl der übertragenen Bytes.

- Sie werden in den ITU-T X.691 und ISO 8825-2 spezifiziert.
- Praktischen Einsatz finden ASN.1 und die BER im Simple Network Management Protocol (SNMP).
- ASN.1 wird zur Beschreibung der SNMP-Pakete verwendet. Die Kodierung für den Transport wird mit Hilfe der BER erstellt.

5.4. Anwendungsschicht

5.4.0.1. Anwendungsschicht

Die Anwendungsschicht (Application Layer) stellt Anwendungen spezifische Dienste sowie Zugang zu Diensten darunter liegender Schichten zur Verfügung. Protokolle der Anwendungsschicht können Teil einer Anwendung sein (z. B. bei Webserver oder Webbrowser).
Beispiele für Dienste der Anwendungsschicht:

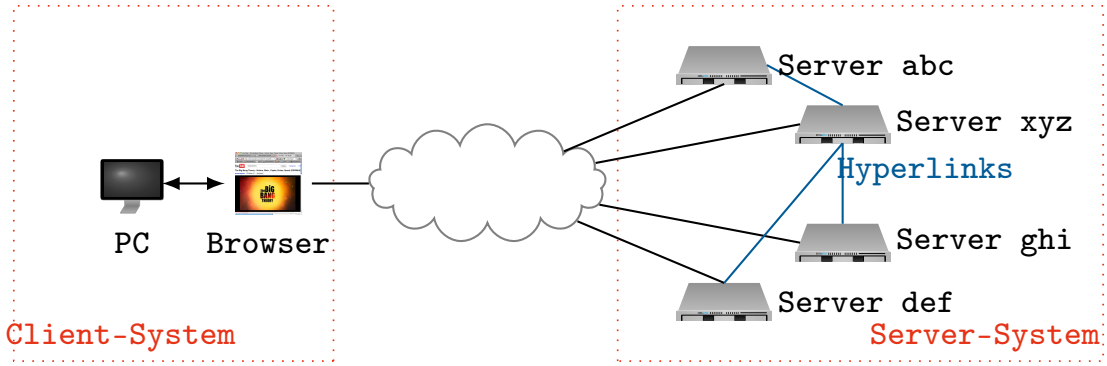
- Namensauflösung (DNS)
- Dateitransfer (HTTP, FTP)
- Nachrichtentransfer (SMTP, POP, IMAP, NNTP)
- Entfernte Anmeldung (Telnet, SSH)
- Netzmanagement (SNMP)

Im Folgenden behandeln wir DNS, HTTP und SMTP als wichtige Protokolle der Anwendungsschicht etwas genauer.

5.4.1. World Wide Web (WWW)

World Wide Web (WWW)

- Um die Protokolle DNS und HTTP besser einordnen zu können, werfen wir einen kurzen Blick auf das World Wide Web, kurz Web.
- Es handelt sich dabei um das architektonische Rahmenwerk zum Dokumentzugriff im Internet.
- 1994 wurde das World Wide Web Consortium (W3C) mit dem Ziel, Web-Technologien weiterzuentwickeln und zu standardisieren.
- Das WWW begründet sich auf einer Client-Server-Architektur, die im Folgenden kurz dargestellt wird.

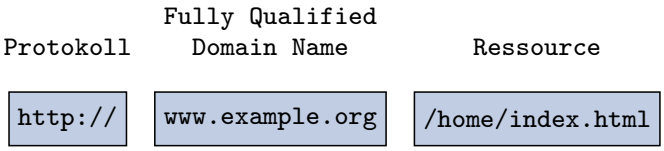


- Zur client-seitigen Darstellung von Web-Seiten (Pages) wird ein Browser verwendet.
- Hyperlinks verknüpfen unterschiedliche Seiten miteinander.

Typischer Ablauf aus Sicht des Clients beim Aufruf einer Web-Seite

- Browser prüft bzw. parst URL (Uniform Resource Locator)

Eine URL besteht im Wesentlichen aus drei Bestandteilen:



- Ermittlung der IP-Adresse mit Hilfe von DNS
- Aufbau einer Verbindung (TCP) über Port 80 zur ermittelten IP-Adresse
- Anfordern der Datei /home/index.html

5.4.2. Domain Name System (DNS)

Domain Name System (DNS)
Motivation:

- Möchte ein Nutzer (Mensch) einen Computer adressieren, will er sich gewöhnlich nicht dessen IP-Adresse merken müssen.
- Stattdessen adressiert man das Ziel i. d. R. mittles eines hierarchisch aufgebauten Namens, z. B. www.google.com.

Domain Name System (DNS)
Motivation:

- Möchte ein Nutzer (Mensch) einen Computer adressieren, will er sich gewöhnlich nicht dessen IP-Adresse merken müssen.
- Stattdessen adressiert man das Ziel i. d. R. mittles eines hierarchisch aufgebauten Namens, z. B. www.google.com.

Domain Name System (DNS):

- Ein DNS-Name (Fully Qualified Domain Name, FQDN) besteht aus einem Hostnamen und einem Suffix:

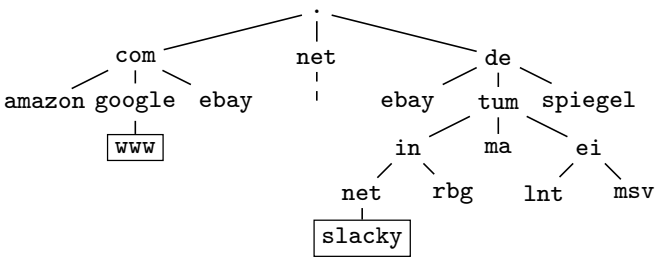
Domain Name System (DNS)
Motivation:

- Möchte ein Nutzer (Mensch) einen Computer adressieren, will er sich gewöhnlich nicht dessen IP-Adresse merken müssen.
- Stattdessen adressiert man das Ziel i. d. R. mittles eines hierarchisch aufgebauten Namens, z. B. www.google.com.

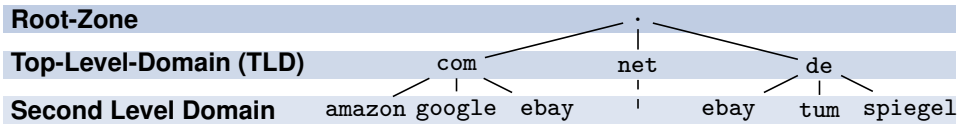
Domain Name System (DNS):

- Ein DNS-Name (Fully Qualified Domain Name, FQDN) besteht aus einem Hostnamen und einem Suffix:

- Das Suffix ist hierarchisch von rechts nach links beginnend bei Root (.) aufgebaut:



Wichtige Begriffe



- Der Namensraum ist in Zonen unterteilt, die unabhängig voneinander administriert werden.
- In jeder Zone gibt es einen primären DNS-Server, welcher
  - eine Liste aller Hosts in dieser Zone und
  - eine Liste mit autoritativen DNS-Servern untergeordneter Domänen

  verwaltet. Änderungen an den Einträgen geschehen nur über den autoritativen DNS-Server.

- Zusätzlich kann es eine Reihe sekundärer Server geben, welche eine Kopie der Zonendatei besitzen ("Zone Transfer", für Lastverteilung und Ausfallsicherheit).
- Die Antworten dieser Server werden als autoritativ bezeichnet, da die Antwort auf Konfigurationsinformation beruht und nicht per DNS von einem anderen Server erhalten wurde. Autoritative Antworten werden von anderen DNS-Servern gecacht.
- Die Cachedauer hängt vom TTL-Wert ab, welcher in jeder Antwort eines Servers enthalten ist.
- Gibt ein DNS-Server einen derartigen gecachten Eintrag weiter, so spricht man von einer nicht-autoritativen Antwort.
- Infolge des Cachings kann es mehrere Stunden (u. U. auch Tage) dauern, bis Änderungen am primären DNS-Server einer Zone im Internet vollständig sichtbar werden.
- Weitere Details zu DNS finden sich u. a. in RFC1035.

In der Zonendatei finden sich DNS Resource Records unterschiedlicher Typen, u. a.:

- SOA – Start Of Authority Enthält Angaben zur Verwaltung der Zone, u. a.
  - Zonenklasse (IN für Internet, HS für Hesiod) Der Hesiod Name Service stammt aus dem Athena-Projekt. Mit ihm lassen sich Informationen aus /etc/passwd etc. zugänglich machen.
    Eine heute übliche Methode besteht darin, diese Information per LDAP zugänglich zu machen.
  - Seriennummer
  - Refresh-Abstand (in Sekunden) für Anfragen nach Änderungen
  - Retry-Abstand zur Wiederholung nicht beantworteter Anfragen
  - Expire-Zeit nach der eine Zone von einem Slave nach einem Zonentransfer-Request als deaktiviert betrachtet wird
  - negativ-Caching-TTL, wenn zu einem angefragten Domainnamen kein Eintrag zugeordnet ist
- A – Hosteintrag IPv4
  Enthält das Mapping zwischen einem Hostnamen und einer IPv4-Adresse
- AAAA – Hosteintrag IPv6
  Enthält das Mapping zwischen einem Hostnamen und einer IPv6-Adresse
- NS – Name Server
  Gibt den FQDN eines DNS-Servers an, welcher für eine (untergeordnete) Domäne autoritativ ist
- CNAME – Common Name
  Ermöglicht es, einem bestehenden Hosteintrag einen weiteren Namen zuzuordnen, z. B. könnte für den Host `typo3.net.in.tum.de.` ein CNAME-Eintrag vorhanden sein, so dass dieser auch unter dem Namen `www.net.in.tum.de.` erreichbar wird.
- MX – Mail Exchanger
  Gibt einen Mailserver (SMTP-Server) für eine Domäne an.

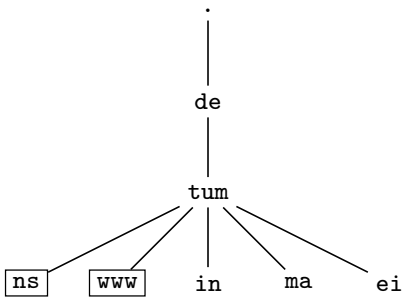
Allgemeine Anmerkungen:

- Einer IP-Adresse können im Prinzip beliebig viele Namen zugewiesen werden.
- Umgekehrt können sich hinter einem Namen auch mehrere Adressen verbergen (einfach mal `nslookup google.com` auf der Kommandozeile ausführen). Warum könnte das sinnvoll sein?
- DNS-Server können auch Reverse-Lookups durchführen, sofern sie dazu konfiguriert wurden (Übersetzung einer IP-Adresse in den/die zugehörigen DNS-Name(n)).

Beispiel: Zonendatei des DNS-Servers `ns.tum.de`

```
$TTL 1W
@  IN  SOA  ns.tum.de. mail.tum.de. (
2007100801 ; serial
28800 ; refresh (8 Stunden)
7200 ; retry (2 Stunden)
604800 ; expire (1 Woche)
39600 ; minimum (11 Stunden)
)

tum.de.         IN  NS  ns.tum.de.
in.tum.de.      IN  NS  ns.in.tum.de.
ma.tum.de.      IN  NS  ns.ma.tum.de.
ei.tum.de.      IN  NS  ns.ei.tum.de.

ns.in.tum.de.   IN  A  <IP von ns.in.tum.de.>
ns.ma.tum.de.   IN  A  <IP von ns.ma.tum.de.>
ns.ei.tum.de.   IN  A  <IP von ns.ei.tum.de.>
ns              IN  A  <IP von ns.tum.de.>
www             IN  A  <IP von www.tum.de.>
```

Ausschnitt aus dem DNS-Namensraum:



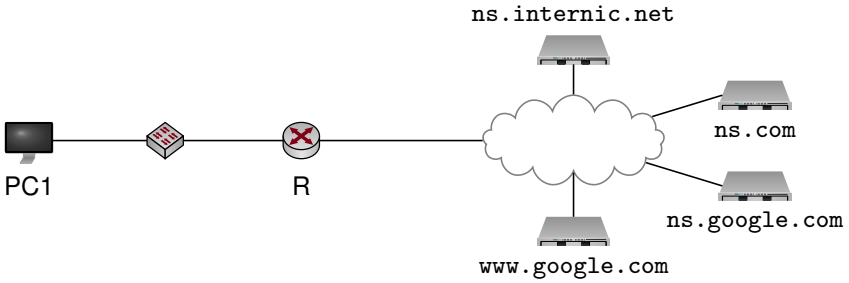
Die Zonendatei von `ns.tum.de.` beinhaltet

- allgemeine Angaben zur Zonendatei (Gültigkeitsdauer, Seriennummer, Refresh-Zeiten usw.),
- eine Liste der autoritativen Namensserver für `tum.de.` selbst sowie alle untergeordneten Domänen, für die `ns.tum.de.` nicht selbst zuständig ist und
- eine Liste mit allen Hostnamen der Domäne(n), für die `ns.tum.de.` selbst zuständig ist.

Wie funktioniert das jetzt aber genau?

Wir nehmen an, PC1 möchte die zum DNS-Namen `www.google.com` gehörende IP-Adresse auflösen. Übliches Setup:

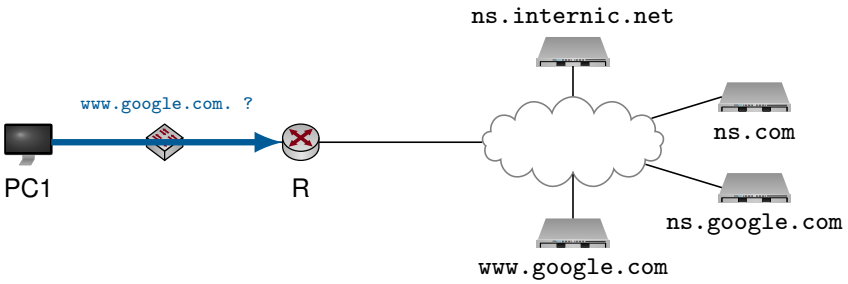
- PC1 ist mit der lokalen IP-Adresse seines Routers R als DNS-Server konfiguriert.
- R selbst ist für keine Zone autoritativ. Seine einzige Aufgabe besteht darin, für Hosts im lokalen Netzwerk die Namensauflösung zu übernehmen.



Wie funktioniert das jetzt aber genau?
Wir nehmen an, PC1 möchte die zum DNS-Namen `www.google.com` gehörende IP-Adresse auflösen. Übliches Setup:

- PC1 ist mit der lokalen IP-Adresse seines Routers R als DNS-Server konfiguriert.

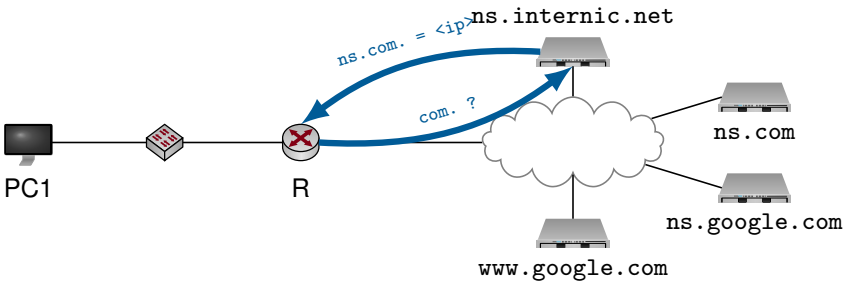
- R selbst ist für keine Zone autoritativ. Seine einzige Aufgabe besteht darin, für Hosts im lokalen Netzwerk die Namensauflösung zu übernehmen.



- PC1 schickt an R einen DNS-Request, um `www.google.com` aufzulösen.

Wie funktioniert das jetzt aber genau?
Wir nehmen an, PC1 möchte die zum DNS-Namen `www.google.com` gehörende IP-Adresse auflösen. Übliches Setup:

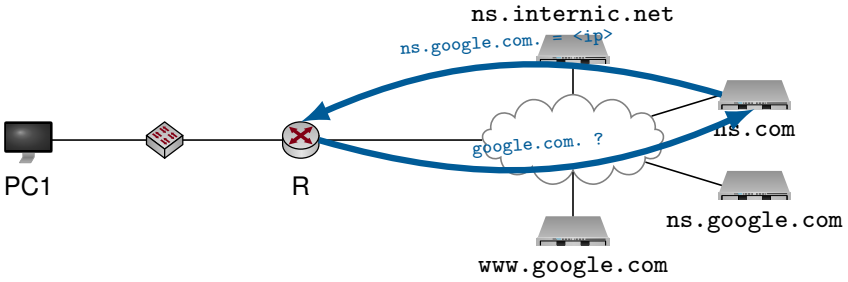
- PC1 ist mit der lokalen IP-Adresse seines Routers R als DNS-Server konfiguriert.
- R selbst ist für keine Zone autoritativ. Seine einzige Aufgabe besteht darin, für Hosts im lokalen Netzwerk die Namensauflösung zu übernehmen.



- PC1 schickt an R einen DNS-Request, um `www.google.com` aufzulösen.
- R kennt weder `www.google.com`, noch `google.com` oder `com` selbst und fragt deshalb bei einem der Root-Server an, welche DNS-Server für die Zone `com.` verantwortlich sind.

Wie funktioniert das jetzt aber genau?
Wir nehmen an, PC1 möchte die zum DNS-Namen `www.google.com` gehörende IP-Adresse auflösen. Übliches Setup:

- PC1 ist mit der lokalen IP-Adresse seines Routers R als DNS-Server konfiguriert.
- R selbst ist für keine Zone autoritativ. Seine einzige Aufgabe besteht darin, für Hosts im lokalen Netzwerk die Namensauflösung zu übernehmen.

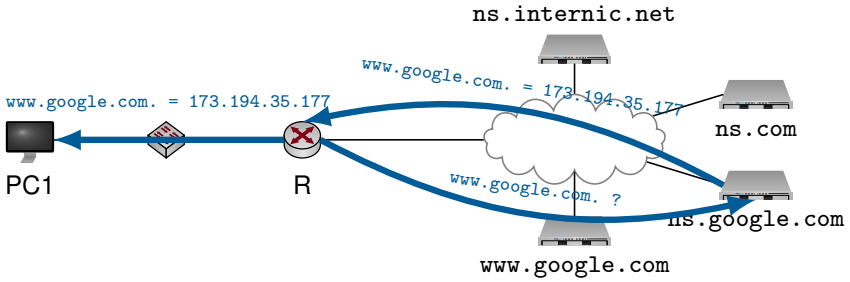


- PC1 schickt an R einen DNS-Request, um `www.google.com` aufzulösen.
- R kennt weder `www.google.com`, noch `google.com` oder `com` selbst und fragt deshalb bei einem der Root-Server an, welche DNS-Server für die Zone `com.` verantwortlich sind.

- Anschließend fragt R bei `ns.com`, wer für `google.com` verantwortlich ist.

Wie funktioniert das jetzt aber genau?
Wir nehmen an, PC1 möchte die zum DNS-Namen `www.google.com` gehörende IP-Adresse auflösen. Übliches Setup:

- PC1 ist mit der lokalen IP-Adresse seines Routers R als DNS-Server konfiguriert.
- R selbst ist für keine Zone autoritativ. Seine einzige Aufgabe besteht darin, für Hosts im lokalen Netzwerk die Namensauflösung zu übernehmen.



- PC1 schickt an R einen DNS-Request, um `www.google.com` aufzulösen.
- R kennt weder `www.google.com`, noch `google.com` oder `com` selbst und fragt deshalb bei einem der Root-Server an, welche DNS-Server für die Zone `com.` verantwortlich sind.
- Anschließend fragt R bei `ns.com`, wer für `google.com` verantwortlich ist.
- Schließlich wendet sich R an `ns.google.com` um die IP-Adresse von `www.google.com` aufzulösen. Das Ergebnis wird an PC1 weitergeleitet und im Cache von R gespeichert (TTL ist in der Antwort von `ns.google.com` enthalten).

Rekursive Namensauflösung

- PC1 sendet einmal einen Request an seinen DNS-Server und erwartet als Response das Ergebnis der Namensauflösung.
- Welche Schritte R unternehmen muss (oder unternimmt), um den Request zu beantworten, weiß PC1 nicht.
- Zwischen PC1 und R werden also nur zwei Pakete ausgetauscht.

Iterative Namensauflösung

- Sofern R nicht bereits Teile des angefragten Namens in seinem Cache hat (z. B. den ganzen FQDN oder einen autoritativen DNS-Server für `google.com`), wird der FQDN Schritt für Schritt (iterativ) beginnend bei der Wurzel aufgelöst.
- Die IP-Adressen der DNS-Rootserver hat jeder DNS-Server (also auch der minimalistische Server auf R) gespeichert. Diese bezeichnet man als Root Hints.
- Prinzipiell wäre es natürlich auch möglich (und ist eher die Regel als die Ausnahme), dass R selbst zur rekursiven Namensauflösung konfiguriert ist. In diesem Fall würde R den Request von PC1 einfach an seinen DNS-Server weiterleiten, z. B. den DNS-Server eines Service Providers wie der Deutschen Telekom.

5.4.3. HyperText Transfer Protocol (HTTP)

HyperText Transfer Protocol (HTTP)

- Als Standardübertragungsprotokoll im Web wird definiert, welche Anfragen ein Client senden und wie der Server antworten darf.
- Es handelt sich um ein ASCII-Anwendungsprotokoll, welches im RFC 2616 in der Version HTTP 1.1 definiert ist.
- Es setzt auf eine TCP-Verbindung auf und verwendet standardmäßig den Port 80.
- In der Version HTTP 1.0 wird vom Server nach jeder beantworteten Anfrage die TCP-Verbindung abgebaut.

HyperText Transfer Protocol (HTTP)

- Als Standardübertragungsprotokoll im Web wird definiert, welche Anfragen ein Client senden und wie der Server antworten darf.
- Es handelt sich um ein ASCII-Anwendungsprotokoll, welches im RFC 2616 in der Version HTTP 1.1 definiert ist.
- Es setzt auf eine TCP-Verbindung auf und verwendet standardmäßig den Port 80.
- In der Version HTTP 1.0 wird vom Server nach jeder beantworteten Anfrage die TCP-Verbindung abgebaut.

Welche Nachteile ergeben sich, wenn eine Web-Seite mit vielen Grafiken angefragt wird?
HyperText Transfer Protocol (HTTP)

- Als Standardübertragungsprotokoll im Web wird definiert, welche Anfragen ein Client senden und wie der Server antworten darf.
- Es handelt sich um ein ASCII-Anwendungsprotokoll, welches im RFC 2616 in der Version HTTP 1.1 definiert ist.
- Es setzt auf eine TCP-Verbindung auf und verwendet standardmäßig den Port 80.
- In der Version HTTP 1.0 wird vom Server nach jeder beantworteten Anfrage die TCP-Verbindung abgebaut.

Welche Nachteile ergeben sich, wenn eine Web-Seite mit vielen Grafiken angefragt wird?

- Jede einzelne Grafik wird mit einer eigenen TCP-Verbindung übertragen.
- <4-> **Verbesserung:** HTTP 1.1 verwendet sog. persistent connections (mit typischem timeout 5-15 sec).

HyperText Transfer Protocol (HTTP)

- Als Standardübertragungsprotokoll im Web wird definiert, welche Anfragen ein Client senden und wie der Server antworten darf.
- Es handelt sich um ein ASCII-Anwendungsprotokoll, welches im RFC 2616 in der Version HTTP 1.1 definiert ist.
- Es setzt auf eine TCP-Verbindung auf und verwendet standardmäßig den Port 80.
- In der Version HTTP 1.0 wird vom Server nach jeder beantworteten Anfrage die TCP-Verbindung abgebaut.

Welche Nachteile ergeben sich, wenn eine Web-Seite mit vielen Grafiken angefragt wird?

- Jede einzelne Grafik wird mit einer eigenen TCP-Verbindung übertragen.
- <4-> **Verbesserung:** HTTP 1.1 verwendet sog. persistent connections (mit typischem timeout 5-15 sec).

HyperText Transfer Protocol (HTTP)

- Als Standardübertragungsprotokoll im Web wird definiert, welche Anfragen ein Client senden und wie der Server antworten darf.
- Es handelt sich um ein ASCII-Anwendungsprotokoll, welches im RFC 2616 in der Version HTTP 1.1 definiert ist.
- Es setzt auf eine TCP-Verbindung auf und verwendet standardmäßig den Port 80.
- In der Version HTTP 1.0 wird vom Server nach jeder beantworteten Anfrage die TCP-Verbindung abgebaut.

Welche Nachteile ergeben sich, wenn eine Web-Seite mit vielen Grafiken angefragt wird?

- Jede einzelne Grafik wird mit einer eigenen TCP-Verbindung übertragen.
- <4-> **Verbesserung:** HTTP 1.1 verwendet sog. persistent connections (mit typischem timeout 5-15 sec).
- Im Folgenden werden HTTP-Requests und HTTP-Responses betrachtet.
- Anhand eines Beispiels wird der Protokollablauf verdeutlicht.

HTTP - Request

- Eine Anfrage des Clients an den Server wird als HTTP-Request bezeichnet.
- Ein HTTP-Request besteht aus der verwendeten Methode und der betreffenden URL.

Bezeichnung	Beschreibung
GET	Der Client fordert den Server auf, eine bestimmte Ressource zu versenden
HEAD	Der Client fordert den Server auf, den Header eine bestimmte Ressource zu versenden
PUT	Das Schreiben einer Ressource wird angefordert
POST	Es werden neue Daten an die benannte Ressource angehängt
DELETE	Es wird angefordert eine Ressource zu löschen

- Ein Request-Header kann zusätzliche Informationen enthalten.

Header	Beschreibung
User-Agent	Browser- und Plattforminformationen
Accept	durch den Client unterstützte MIME-Typen, z. B. *text/html*
Accept-Charset	durch den Client unterstützte Zeichensätze, z. B. *Unicode*
Accept-Encoding	durch den Client unterstützte Kompressionsverfahren, z. B. *gzip*
Accept-Language	natürliche Sprache, z. B. *english*
Host	DNS-Name des Servers
Authorization	Anmeldeinformationen des Clients
Cookies	vom Server erhaltene Cookies

HTTP - Response

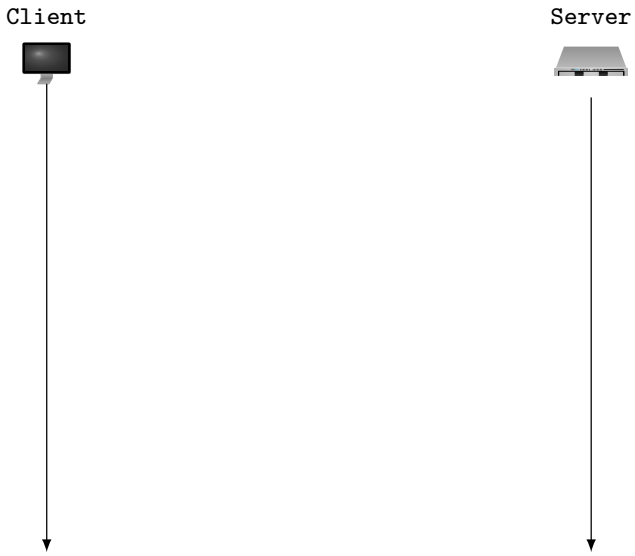
- Ein HTTP-Response beginnt mit der Statuszeile mit einem entsprechenden Status-Code.

Code	Bedeutung	Beispiel
1xx	Information	102 = *Processing*, zur Verhinderung eines Timeouts
2xx	Erfolg	200 = *OK*, die Anfrage wurde erfolgreich bearbeitet
3xx	Umleitung	301 = *Moved Permanently*, die alte Adresse ist nicht länger gültig
4xx	Client-Fehler	400 = *Bad Request*, fehlerhaft aufgabeute Anfrage Nachricht
5xx	Server-Fehler	503 = *Service Unavailable*, Server steht temporär nicht zur Verfügung

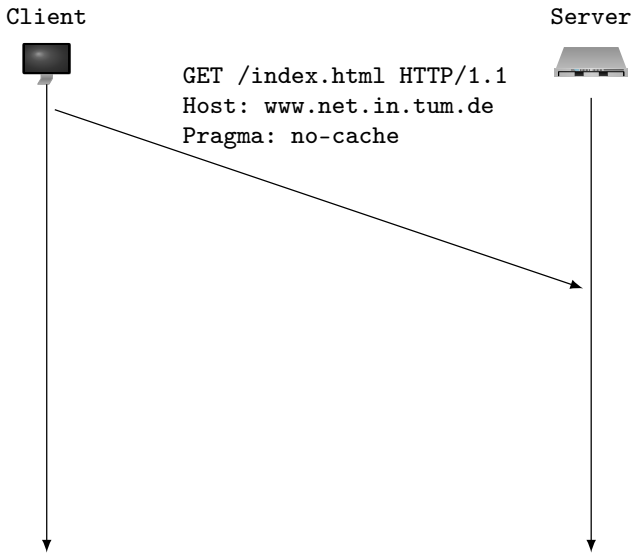
- Nach der Statuszeile folgt der Response-Header, sowie ggf. die angeforderte Ressource.

Response-Header	Beschreibung
Server	Informationen über den Server
Content-Lenght	Seitenlänge in Byte
Content-Encoding	Kodierungsinformationen
Content-Language	natürliche Sprache der Seite
Content-Type	MIME-Type der Seite
Last-Modified	Letztes Änderungsdatum der Seite
Location	Anforderungen müssen an gegeben Server gesendet werden
Accept-Ranges	Server akzeptiert Bereichsanfragen in Byte
Set-Cookies	Aufforderung an den Client ein Cookie zu speichern

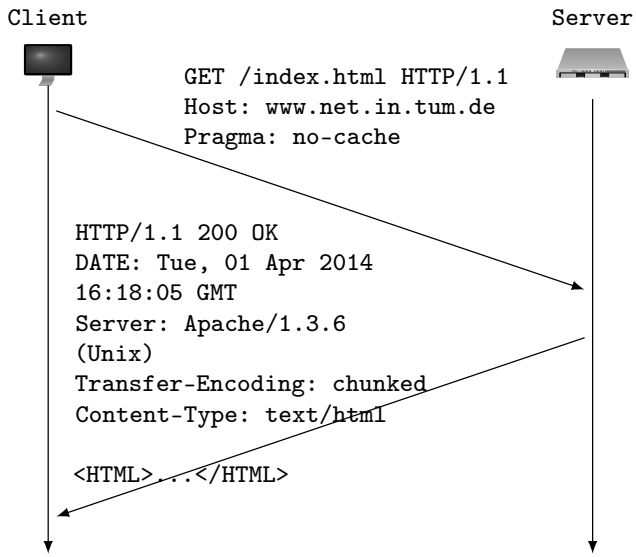
Beispiel



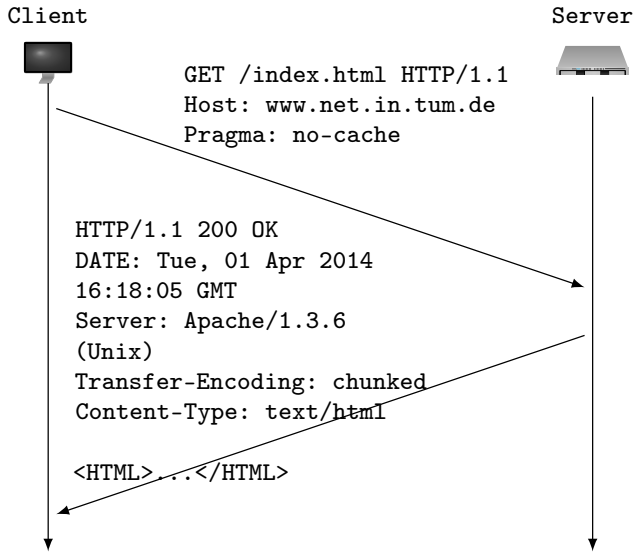
Beispiel



Beispiel



Beispiel



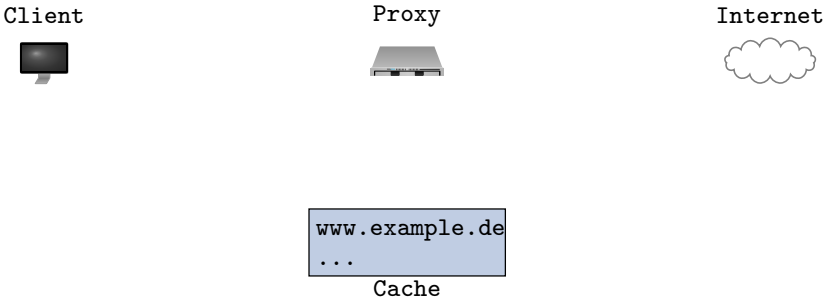
- Die Header-Felder Date und Upgrade (Wechsel des Protokolls) können sowohl im Request als auch im Response verwendet werden.

Gibt es Möglichkeiten Ressourcen schneller zu laden?

- Optimierungen können sowohl auf Seiten des Client, als auch auf Seiten der Server durchgeführt werden.
- Hierfür stehen verschiedene Möglichkeiten zur Verfügung, u. a. Caching mit Hilfe von Proxies, Replikation von Servern (Mirroring) oder sog. Content Delivery Networks.

Definition (Proxy):

Ein Proxy-Server wird zwischen den Client und den Servern postiert und prüft, ob vom Client angefragte Seiten im Cache verfügbar sind oder vom Server geladen werden müssen.

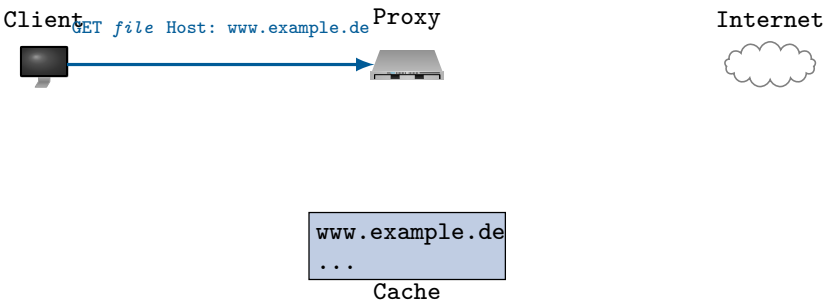


Gibt es Möglichkeiten Ressourcen schneller zu laden?

- Optimierungen können sowohl auf Seiten des Client, als auch auf Seiten der Server durchgeführt werden.
- Hierfür stehen verschiedene Möglichkeiten zur Verfügung, u. a. Caching mit Hilfe von Proxies, Replikation von Servern (Mirroring) oder sog. Content Delivery Networks.

Definition (Proxy):

Ein Proxy-Server wird zwischen den Client und den Servern postiert und prüft, ob vom Client angefragte Seiten im Cache verfügbar sind oder vom Server geladen werden müssen.

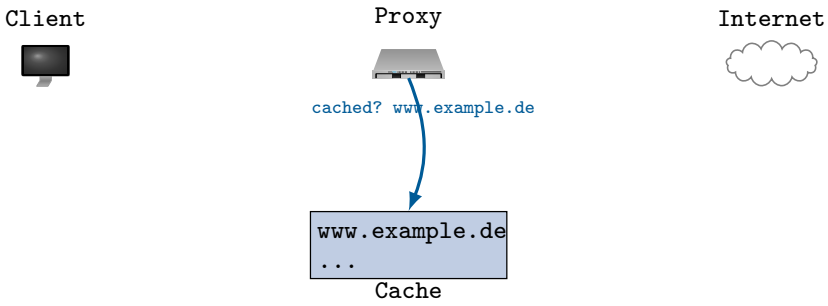


Gibt es Möglichkeiten Ressourcen schneller zu laden?

- Optimierungen können sowohl auf Seiten des Client, als auch auf Seiten der Server durchgeführt werden.
- Hierfür stehen verschiedene Möglichkeiten zur Verfügung, u. a. Caching mit Hilfe von Proxies, Replikation von Servern (Mirroring) oder sog. Content Delivery Networks.

Definition (Proxy):

Ein Proxy-Server wird zwischen den Client und den Servern postiert und prüft, ob vom Client angefragte Seiten im Cache verfügbar sind oder vom Server geladen werden müssen.

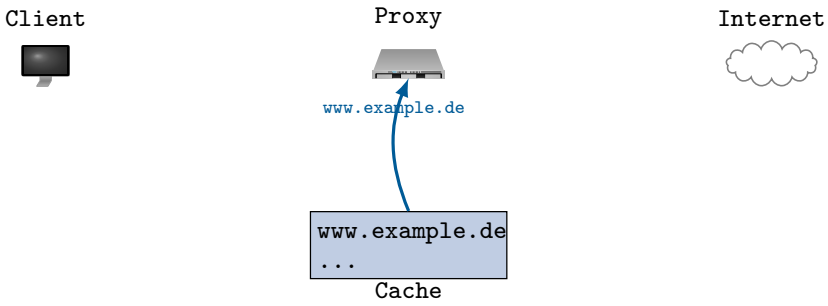


Gibt es Möglichkeiten Ressourcen schneller zu laden?

- Optimierungen können sowohl auf Seiten des Client, als auch auf Seiten der Server durchgeführt werden.
- Hierfür stehen verschiedene Möglichkeiten zur Verfügung, u. a. Caching mit Hilfe von Proxies, Replikation von Servern (Mirroring) oder sog. Content Delivery Networks.

Definition (Proxy):

Ein Proxy-Server wird zwischen den Client und den Servern postiert und prüft, ob vom Client angefragte Seiten im Cache verfügbar sind oder vom Server geladen werden müssen.

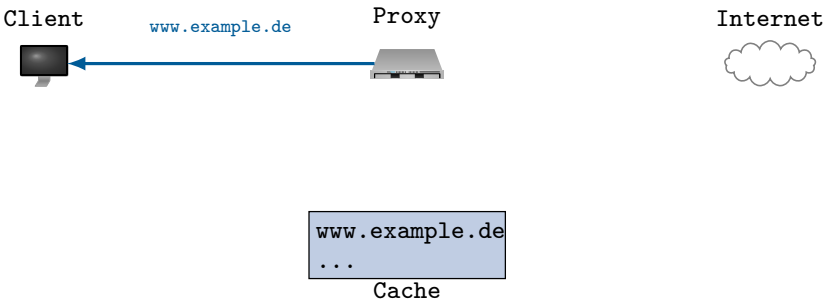


Gibt es Möglichkeiten Ressourcen schneller zu laden?

- Optimierungen können sowohl auf Seiten des Client, als auch auf Seiten der Server durchgeführt werden.
- Hierfür stehen verschiedene Möglichkeiten zur Verfügung, u. a. Caching mit Hilfe von Proxies, Replikation von Servern (Mirroring) oder sog. Content Delivery Networks.

Definition (Proxy):

Ein Proxy-Server wird zwischen den Client und den Servern postiert und prüft, ob vom Client angefragte Seiten im Cache verfügbar sind oder vom Server geladen werden müssen.

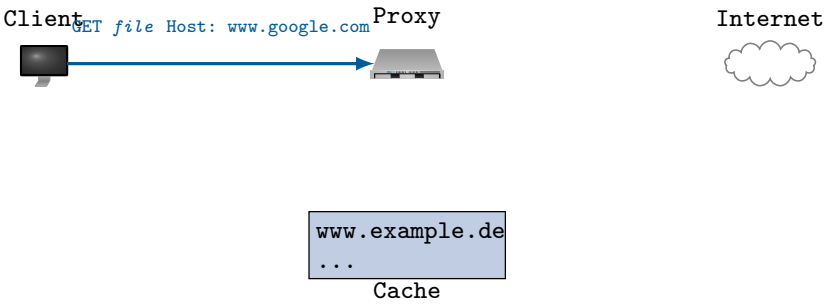


Gibt es Möglichkeiten Ressourcen schneller zu laden?

- Optimierungen können sowohl auf Seiten des Client, als auch auf Seiten der Server durchgeführt werden.
- Hierfür stehen verschiedene Möglichkeiten zur Verfügung, u. a. Caching mit Hilfe von Proxies, Replikation von Servern (Mirroring) oder sog. Content Delivery Networks.

Definition (Proxy):

Ein Proxy-Server wird zwischen den Client und den Servern postiert und prüft, ob vom Client angefragte Seiten im Cache verfügbar sind oder vom Server geladen werden müssen.

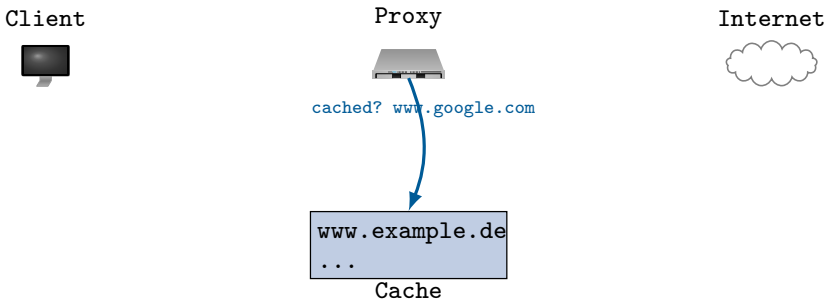


Gibt es Möglichkeiten Ressourcen schneller zu laden?

- Optimierungen können sowohl auf Seiten des Client, als auch auf Seiten der Server durchgeführt werden.
- Hierfür stehen verschiedene Möglichkeiten zur Verfügung, u. a. Caching mit Hilfe von Proxies, Replikation von Servern (Mirroring) oder sog. Content Delivery Networks.

Definition (Proxy):

Ein Proxy-Server wird zwischen den Client und den Servern postiert und prüft, ob vom Client angefragte Seiten im Cache verfügbar sind oder vom Server geladen werden müssen.

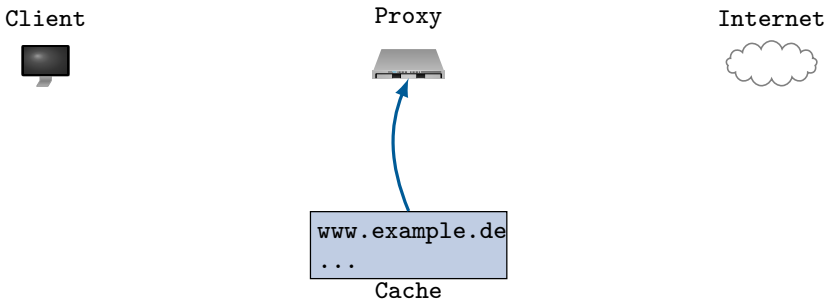


Gibt es Möglichkeiten Ressourcen schneller zu laden?

- Optimierungen können sowohl auf Seiten des Client, als auch auf Seiten der Server durchgeführt werden.
- Hierfür stehen verschiedene Möglichkeiten zur Verfügung, u. a. Caching mit Hilfe von Proxies, Replikation von Servern (Mirroring) oder sog. Content Delivery Networks.

Definition (Proxy):

Ein Proxy-Server wird zwischen den Client und den Servern postiert und prüft, ob vom Client angefragte Seiten im Cache verfügbar sind oder vom Server geladen werden müssen.

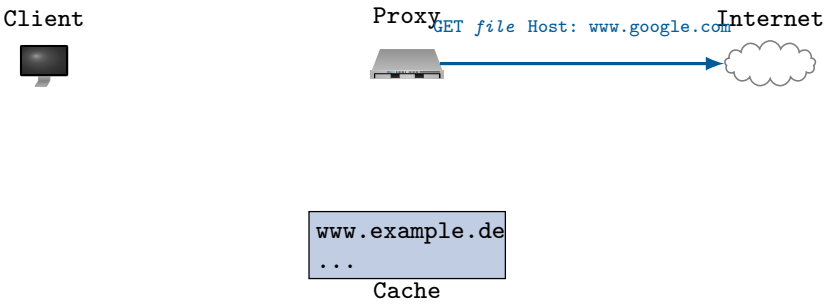


Gibt es Möglichkeiten Ressourcen schneller zu laden?

- Optimierungen können sowohl auf Seiten des Client, als auch auf Seiten der Server durchgeführt werden.
- Hierfür stehen verschiedene Möglichkeiten zur Verfügung, u. a. Caching mit Hilfe von Proxies, Replikation von Servern (Mirroring) oder sog. Content Delivery Networks.

Definition (Proxy):

Ein Proxy-Server wird zwischen den Client und den Servern postiert und prüft, ob vom Client angefragte Seiten im Cache verfügbar sind oder vom Server geladen werden müssen.

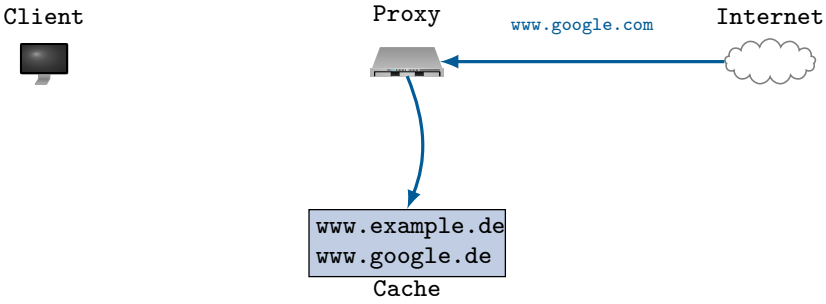


Gibt es Möglichkeiten Ressourcen schneller zu laden?

- Optimierungen können sowohl auf Seiten des Client, als auch auf Seiten der Server durchgeführt werden.
- Hierfür stehen verschiedene Möglichkeiten zur Verfügung, u. a. Caching mit Hilfe von Proxies, Replikation von Servern (Mirroring) oder sog. Content Delivery Networks.

Definition (Proxy):

Ein Proxy-Server wird zwischen den Client und den Servern postiert und prüft, ob vom Client angefragte Seiten im Cache verfügbar sind oder vom Server geladen werden müssen.

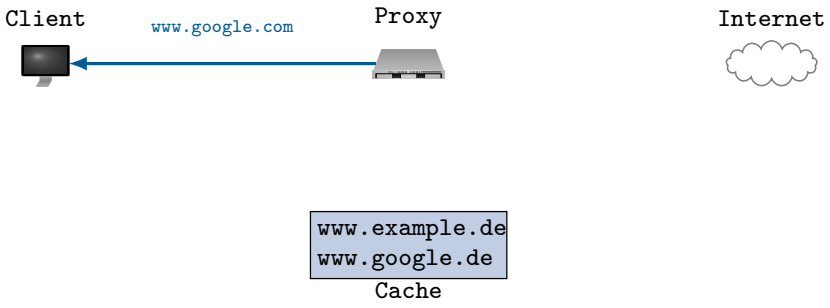


Gibt es Möglichkeiten Ressourcen schneller zu laden?

- Optimierungen können sowohl auf Seiten des Client, als auch auf Seiten der Server durchgeführt werden.
- Hierfür stehen verschiedene Möglichkeiten zur Verfügung, u. a. Caching mit Hilfe von Proxies, Replikation von Servern (Mirroring) oder sog. Content Delivery Networks.

Definition (Proxy):

Ein Proxy-Server wird zwischen den Client und den Servern postiert und prüft, ob vom Client angefragte Seiten im Cache verfügbar sind oder vom Server geladen werden müssen.



5.4.4. Electronic-Mail (e-Mail)

Electronic-Mail (e-Mail)

- Die ersten Vorschläge zur Standardisierung wurden bereits 1982 in den RFCs 821 (Übertragungsprotokoll) und 822 (Nachrichtenformat) vorgelegt.
- Endgültig zum Standard wurden die entsprechenden Revisionen 2821 und 2822 erklärt.

Definition (e-Mail-Architektur):

Es sind zwei Teilsysteme von Bedeutung: die User-Agents werden von den Benutzern verwendet, um e-Mails zu lesen und zu verwalten. Die Message-Transfer-Agents leiten Nachrichten der Sender bis zum Ziel weiter.

Die folgenden **5 Grundfunktionen** sollten von allen e-Mail-Systemen unterstützt werden:

- Erstellung (Composition)
- Übertragung (Transfer)
- Berichterstellung (Reporting)
- Anzeige (Display)
- Verwertung (Disposition)

e-Mail - Format

e-Mail - Format

```
MAIL FROM: <foo@net.in.tum.de>                                  Envelope
RCPT TO: <ba@net.in.tum.de>                                      für MTA
```

- Der Envelope enthält alle Informationen für den Transport der e-Mail zum Empfänger.

e-Mail - Format

```
MAIL FROM: <foo@net.in.tum.de>                                  Envelope
RCPT TO: <ba@net.in.tum.de>                                      für MTA
To: <ba@online.de>
CC: <andere@online.de>
Bcc: <unsichtbar@online.de>
From: <foo@net.in.tum.de>
Sender: <ghostwriter@net.in.tum.de>
Received: from smtp1.informatik.tu-muenchen.de ([131.159.0.8]) by
(...) for <ba@online.de>; Tue, 01 Apr 2014 10:15:04 +0200
Return-Path: <foo@net.in.tum.de>                                  Header
Date: Tue, 01 Apr 2014 10:15:03 +0200                              für UA
Message-ID: <01ab91cf4d82$7b206430$71627390$@net.in.tum.de>
Reply-to: <foo@net.in.tum.de>
In-Reply-To: <99ab91cf4d82$7b206430$71627390$@net.in.tum.de>
References: <99ab91cf4d82$7b206430$71627390$@net.in.tum.de>
<88ab91cf4d82$7b206430$71627390$@net.in.tum.de>
Subject: Re: Re: Status Update
```

- Der Envelope enthält alle Informationen für den Transport der e-Mail zum Empfänger.
- Der Header ist optional und enthält Informationen über den Absender bzw. Empfänger.

e-Mail - Format

```
MAIL FROM: <foo@net.in.tum.de>                                  Envelope
RCPT TO: <ba@net.in.tum.de>                                      für MTA
To: <ba@online.de>
CC: <andere@online.de>
Bcc: <unsichtbar@online.de>
From: <foo@net.in.tum.de>
Sender: <ghostwriter@net.in.tum.de>
Received: from smtp1.informatik.tu-muenchen.de ([131.159.0.8]) by
(...) for <ba@online.de>; Tue, 01 Apr 2014 10:15:04 +0200
Return-Path: <foo@net.in.tum.de>                                  Header
Date: Tue, 01 Apr 2014 10:15:03 +0200                              für UA
Message-ID: <01ab91cf4d82$7b206430$71627390$@net.in.tum.de>
Reply-to: <foo@net.in.tum.de>
In-Reply-To: <99ab91cf4d82$7b206430$71627390$@net.in.tum.de>
References: <99ab91cf4d82$7b206430$71627390$@net.in.tum.de>
<88ab91cf4d82$7b206430$71627390$@net.in.tum.de>
Subject: Re: Re: Status Update                                      Body
Terminbestätigung
Montag bis Mittwoch ...
```

- Der Envelope enthält alle Informationen für den Transport der e-Mail zum Empfänger.
- Der Header ist optional und enthält Informationen über den Absender bzw. Empfänger.

- Der Body beinhaltet den eigentlichen Inhalt.

Multipurpose Internet Mail Extensions (MIME)

- **Problem:** e-Mails wurden als reine Textnachrichten im ASCII-Format entworfen.
- **Lösung:** Erweiterung der Strukturen für den Nachrichtentext und Definition der Kodierungsregeln für nicht-ASCII-Nachrichten mit Hilfe 5 zusätzlicher Header.

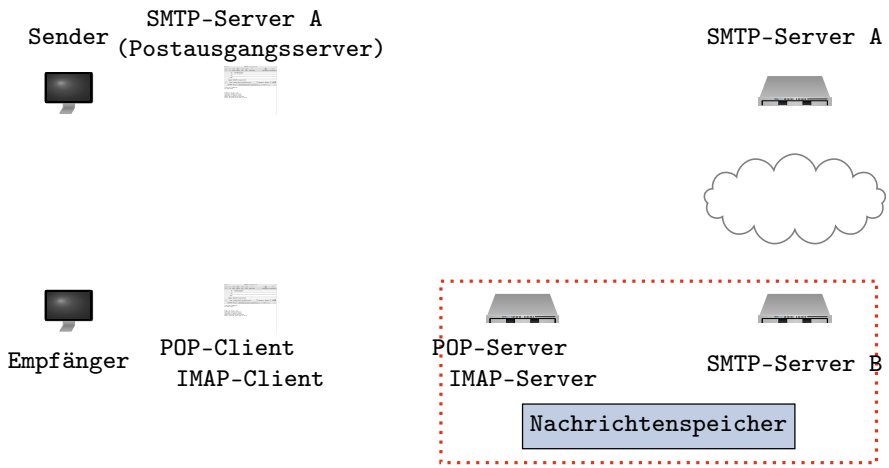
Header	Beschreibung
MIME-Version	verwendete MIME-Version
Content-Description	Beschreibung des Inhalts in ASCII-Zeichen
Content-ID	Eindeutiger Bezeichner im gleichen Format wie die Message-ID
Content-Transfer-Encoding	für die Übertragung verwendete Kodierung (5 Schemata, z. B. base64)
Content-Type	Art des Nachrichteninhalts

Der Content-Type besteht aus der Typangabe sowie dem Untertyp, welche durch einen "/" getrennt werden, z. B. image/gif. Einige Typen sind in der folgenden Tabelle gelistet.

Typ	Untertypen	Beschreibung
Text	plain, enriched	unformatierter bzw. einfach formatierter ASCII-Text
Image	gif, jpeg	Standbild im GIF- oder JPEG-Format
Audio	basic	Audiodateien
Video	mpeg	Video im MPEG-Format

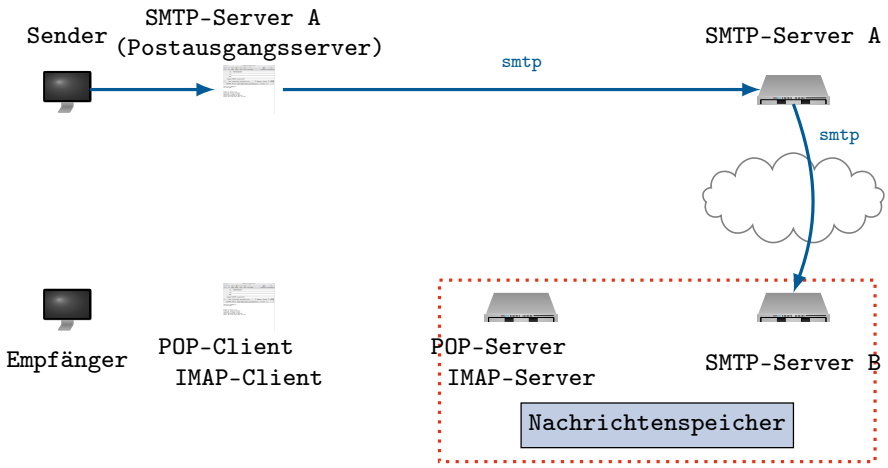
Mailserver-Architektur

- SMTP wird als Protokoll zum Versand von e-Mail zwischen MTAs und von einem UA zum entsprechenden MTA verwendet.
- Nachrichten können dann mittels POP3 oder IMAP vom entprechenden SMTP-Server abgeholt bzw. dort verwaltet werden.



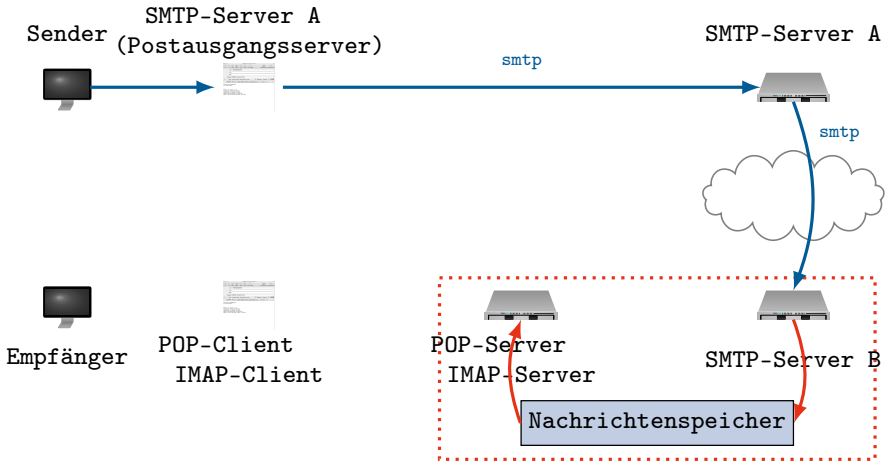
Mailserver-Architektur

- SMTP wird als Protokoll zum Versand von e-Mail zwischen MTAs und von einem UA zum entsprechenden MTA verwendet.
- Nachrichten können dann mittels POP3 oder IMAP vom entprechenden SMTP-Server abgeholt bzw. dort verwaltet werden.



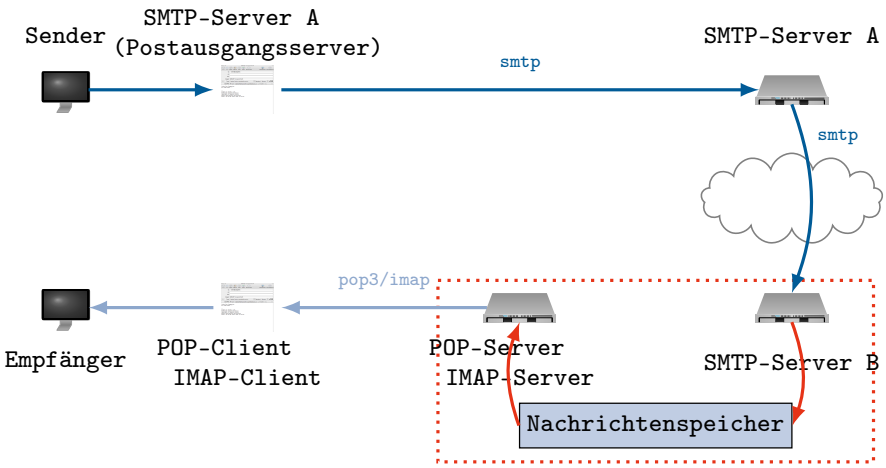
Mailserver-Architektur

- SMTP wird als Protokoll zum Versand von e-Mail zwischen MTAs und von einem UA zum entsprechenden MTA verwendet.
- Nachrichten können dann mittels POP3 oder IMAP vom entprechenden SMTP-Server abgeholt bzw. dort verwaltet werden.



Mailserver-Architektur

- SMTP wird als Protokoll zum Versand von e-Mail zwischen MTAs und von einem UA zum entsprechenden MTA verwendet.
- Nachrichten können dann mittels POP3 oder IMAP vom entprechenden SMTP-Server abgeholt bzw. dort verwaltet werden.



Simple Mail Transfer Protocol (SMTP)

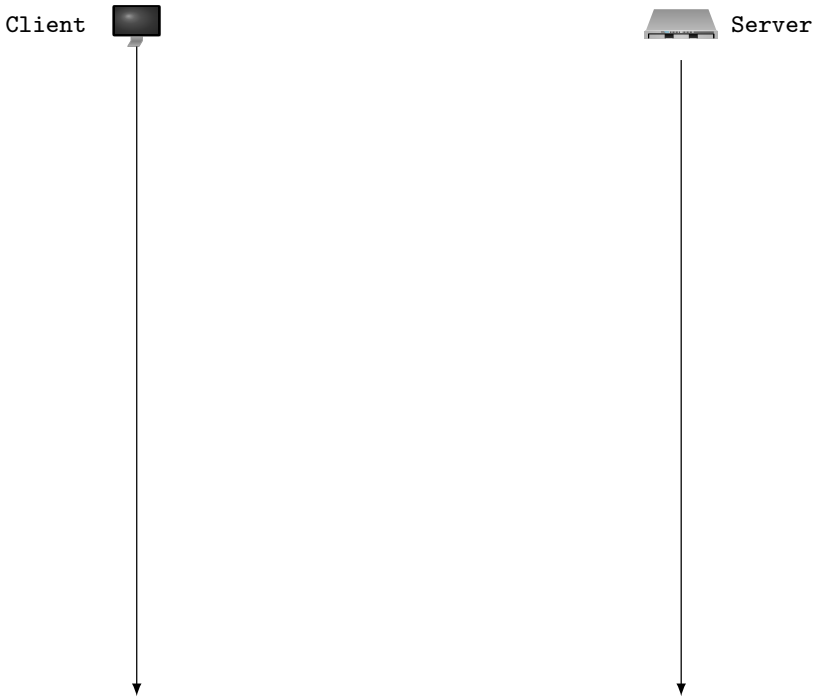
- SMTP ist ein zeichenorientiertes Protokoll und benutzt standardmäßig den Port 25.
- Port 587 wird verwendet, wenn eine Authentifizerung der Benutzer durchgeführt wird. Der Austausch zwischen MTAs findet weiterhin über Port 25 statt.
- Die Kommunikation zwischen SMTP-Entitäten erfolgt über einen einfachen Kommandosatz.

Kommando	Beschreibung
HELO	Start der SMTP Sitzung und Identifikation der Teilnehmer
MAIL	Start der Mailübertragung
RCPT	Angabe der Empfängeradresse, Kommando kann mehrfach ausgeführt werden
DATA	Übermittlung der eigentlichen Nachricht, beendet mittels "."
RSET	Abbruch der Übertragung, Verbindung bleibt bestehen
VRTY oder EXPN	Prüfen der Empfängeradresse
NOOP	Antwort vom Server erzwingen, z. B. um einen Timeout zu vermeiden
QUIT	Beenden der Verbindung

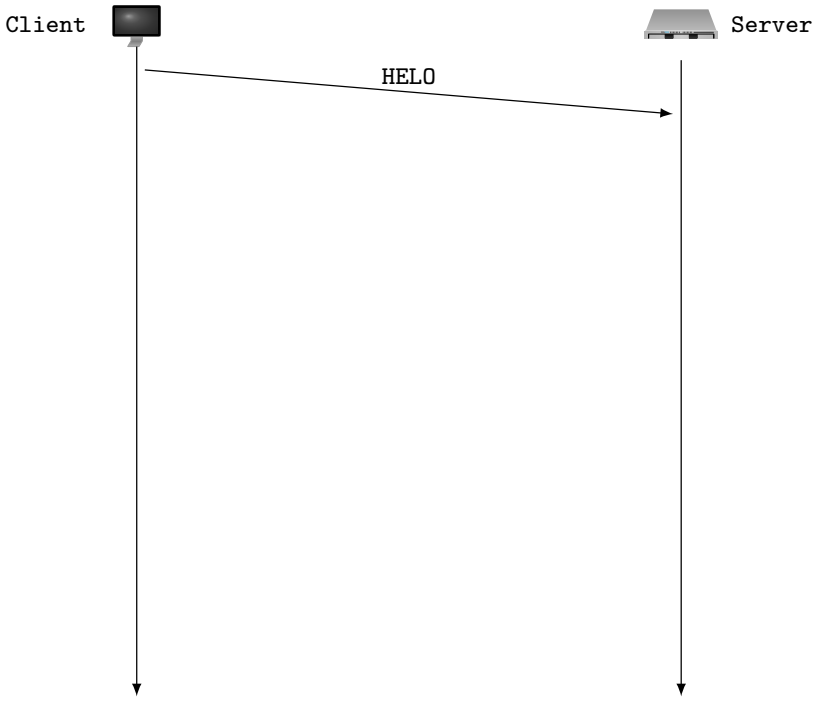
Eine Antwort des Servers besteht aus einem 3-stelligen Statuscode, sowie der Klartextmeldung.

Code	Bedeutung	Code	Bedeutung
250	*OK*, Kommando wurde erfolgreich ausgeführt	354	Starte Mail Input
421	Service nicht verfügbar	451	Ausführungsfehler und Abbruch
503	Falsche Kommandoreihenfolge	554	Transaktion fehlgeschlagen

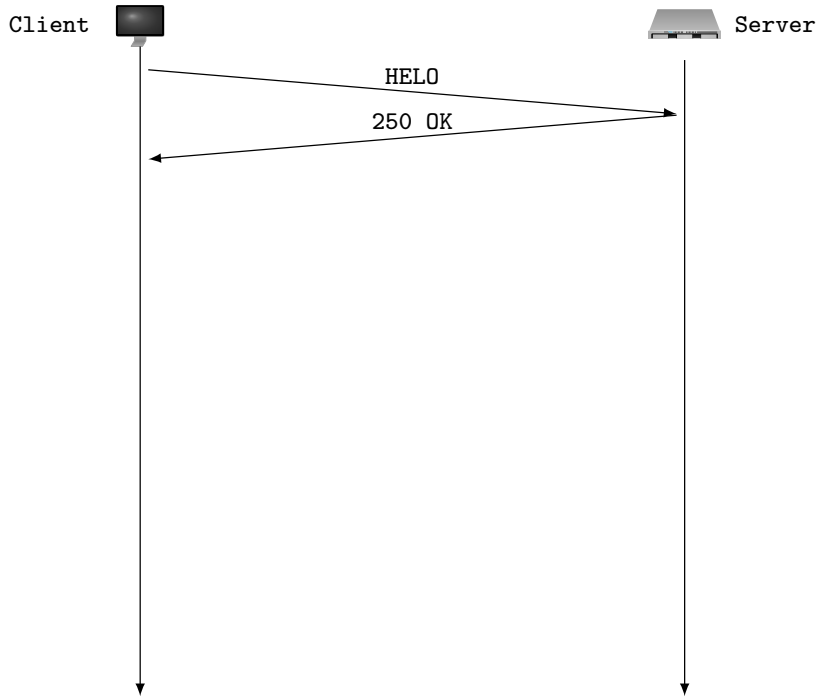
Ablaufbeispiel



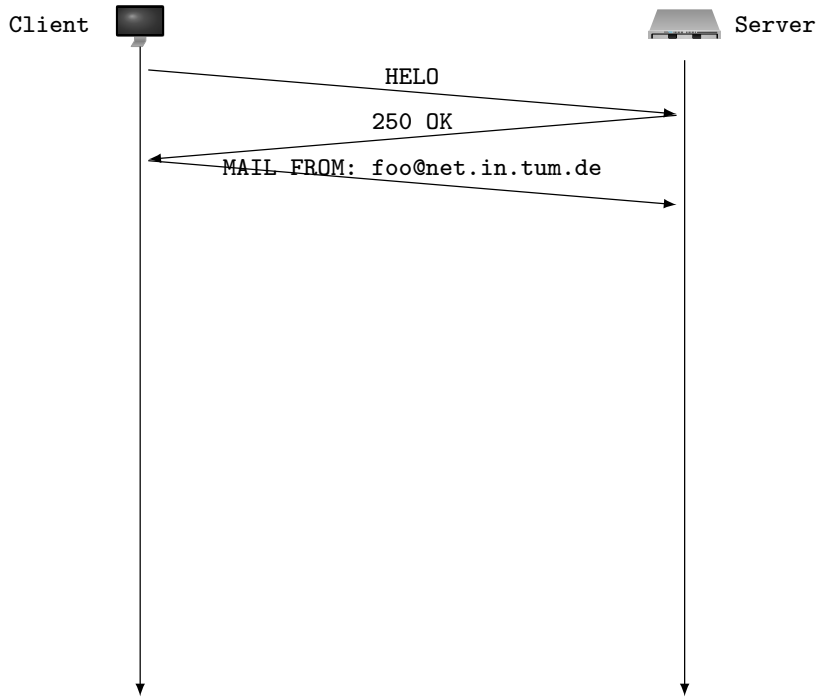
Ablaufbeispiel



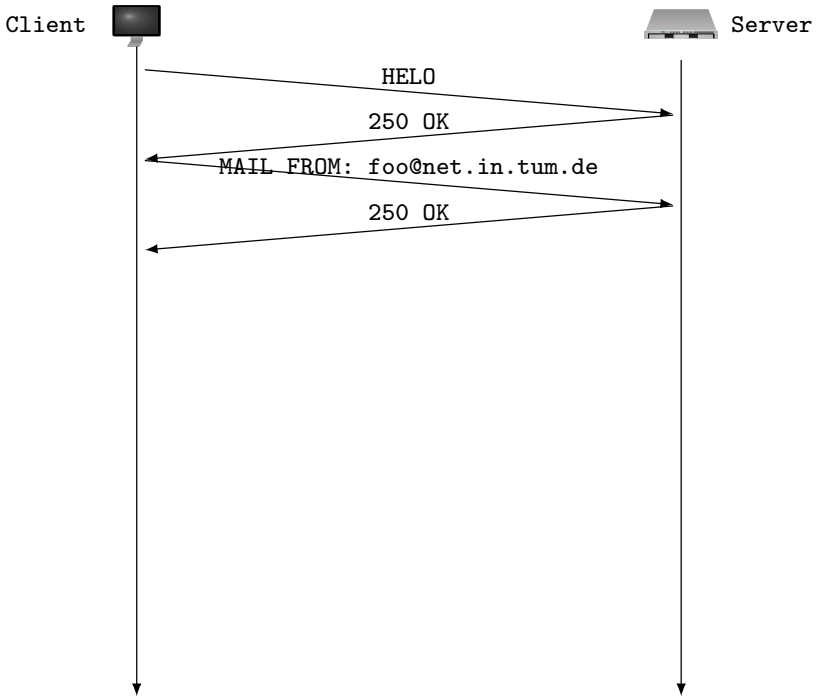
Ablaufbeispiel



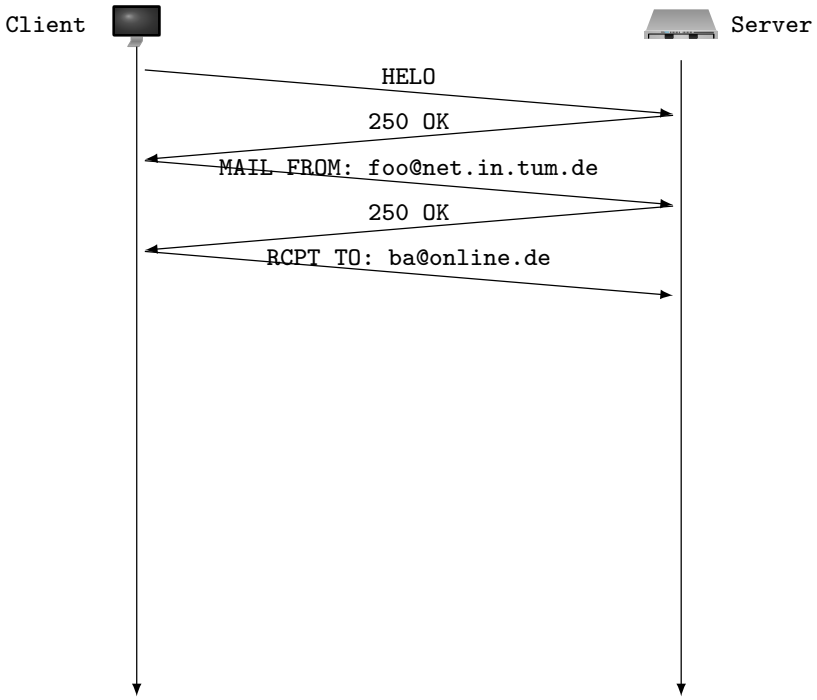
Ablaufbeispiel



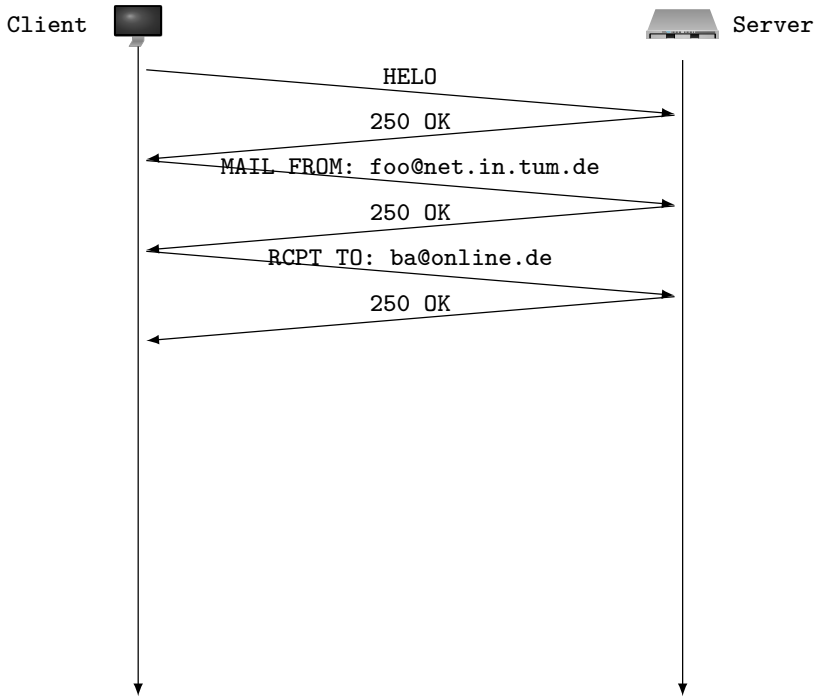
Ablaufbeispiel



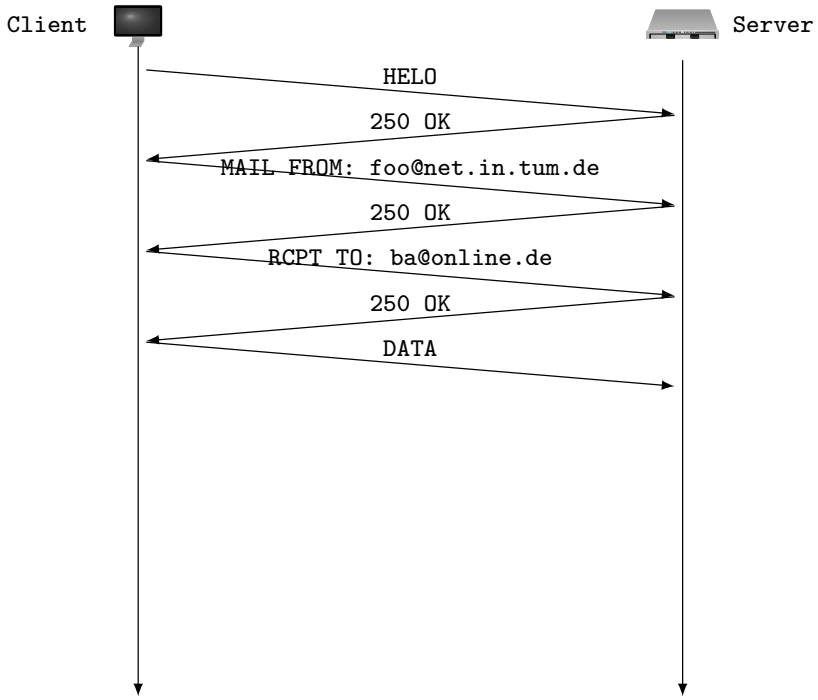
Ablaufbeispiel



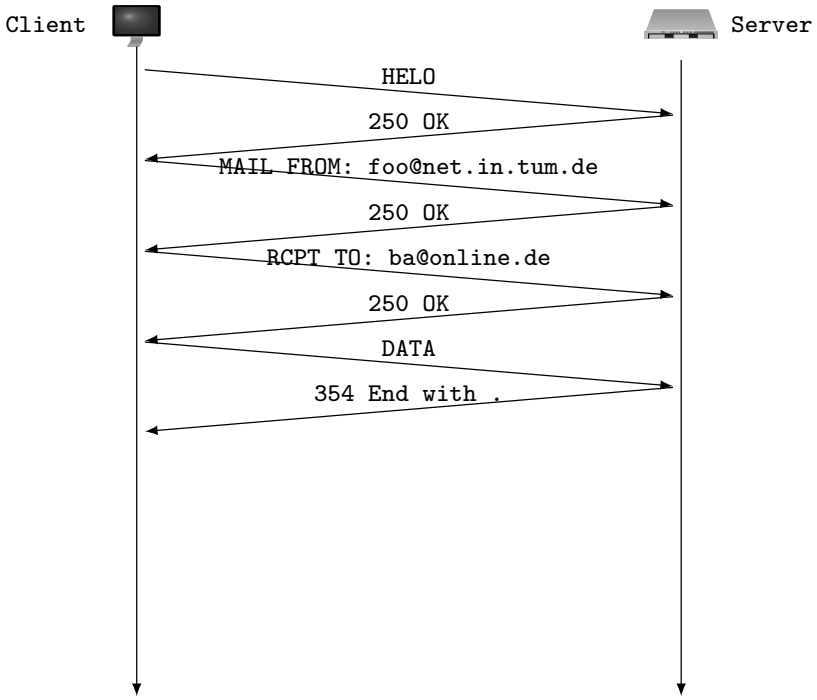
Ablaufbeispiel



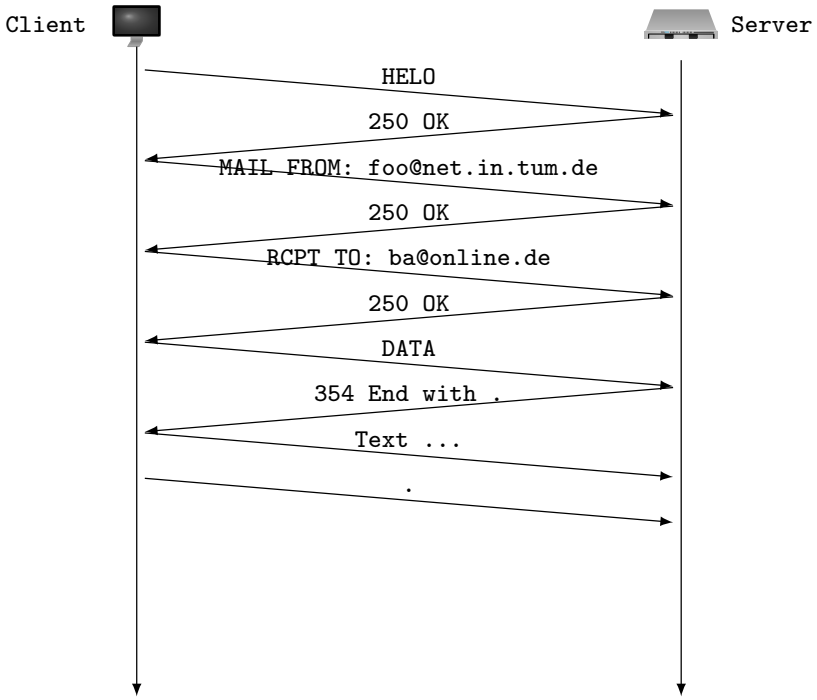
Ablaufbeispiel



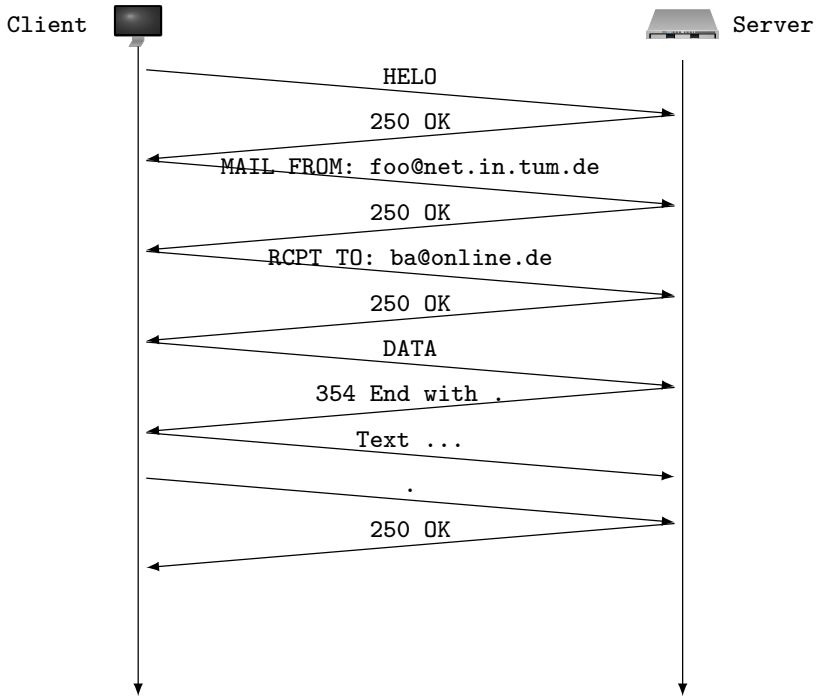
Ablaufbeispiel



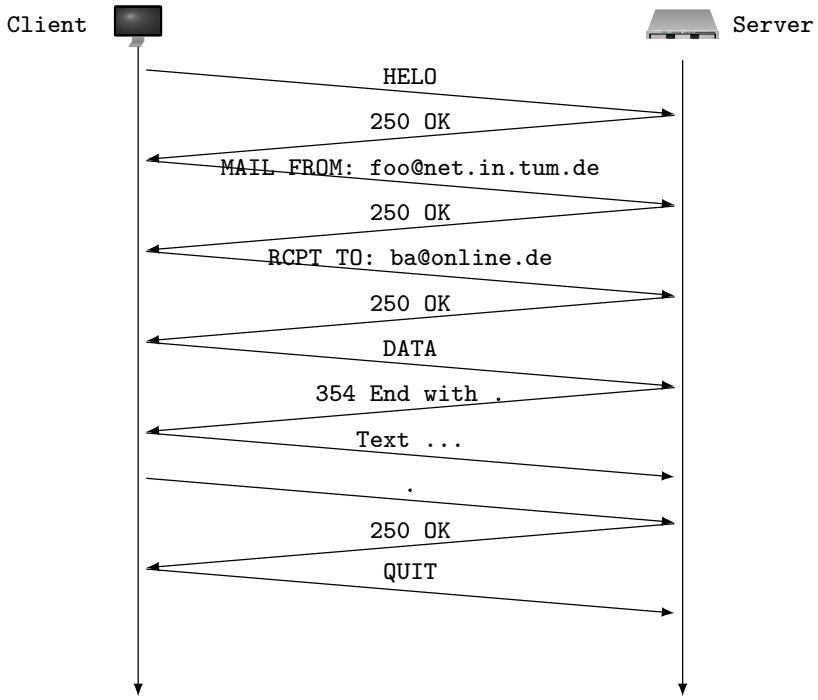
Ablaufbeispiel



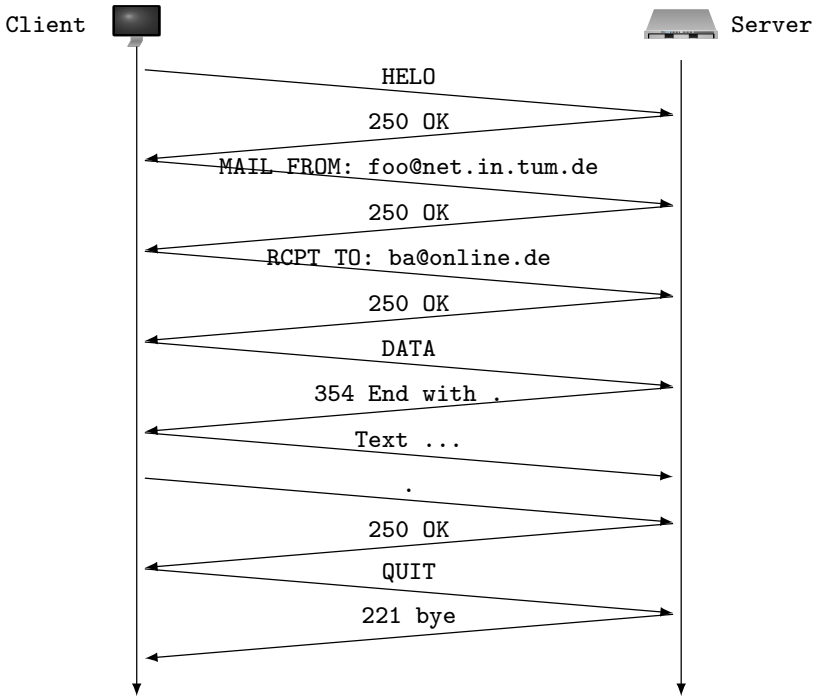
Ablaufbeispiel



Ablaufbeispiel

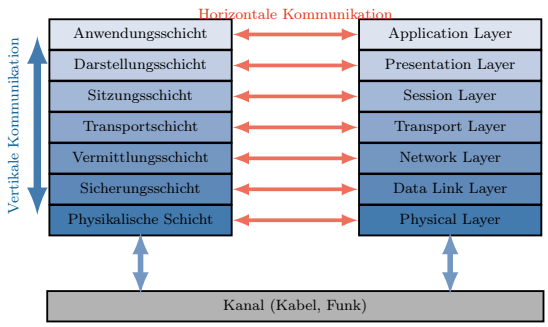


Ablaufbeispiel



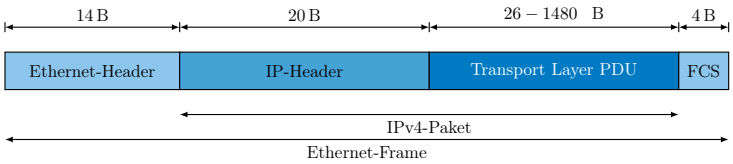
5.5. Zusammenfassung

5.5.0.1. Zusammenfassung: ISO/OSI Modell



- Das ISO/OSI Modell unterteilt den Kommunikationsvorgang in 7 Schichten
- Es spezifiziert, welche Dienste in den verschiedenen Schichten zu erbringen sind
- Es wird jedoch keine Implementation vorgegeben
- Die n-te Schicht des Quellsystems kommuniziert nie direkt mit der n-ten Schicht des Ziels
- Kommunikation erfolgt stets erst vertikal im Stack, dann horizontal im Kanal und abschließend wieder vertikal
- Das Internet ist über einen TCP/IP-Stack realisiert, welcher weniger Schichten umfasst

5.5.0.2. Zusammenfassung: ISO/OSI Modell



- Den Nutzdaten wird auf jeder Ebene ein entsprechender Header vorangestellt
- Die tatsächlich transportierten Daten beinhalten damit für jede aktive Schicht einen Header

5.5.0.3. Zusammenfassung: Schichten

Physikalische Schicht Der von der ersten Schicht angebotene Dienst ist die Punkt-zu-Punkt-Übermittlung von reinen Bits in Form von physikalisch messbaren Signalen.

- Generierung von Signales aus Bits (Impulsformung), welche auf Zielseite wiedererkannt werden müssen (Detektion)
- Auftragen der Signale auf eine Trägerwelle (Modulation)
- Als Trägermedium werden i. d. R. elektromagnetische Wellen genutzt
- Teilweise ausgleichen der inhärenten Unzuverlässigkeit des Kanals mithilfe einer Kanalkodierung

Sicherungsschicht Die Sicherungsschicht übernimmt die Abstraktion eines physischen Kanals auf eine logische Direktverbindung.

- Erkennen von Übertragungsfehlern, nach Möglichkeit Korrektur
- Verhindern einer Überforderung des Zielsystems bei der Kommunikation (Flusskontrolle)
- Regelung des Medienzugriffs auf ein gemeinsam genutztes Kommunikationssystem

5.5.0.4. Zusammenfassung: Schichten

Vermittlungsschicht Die Vermittlungsschicht verbindet Systeme über beliebig viele Direktverbindungen hinweg.

- Ermöglichung von Kommunikation über Direktverbindungen und Subnetze
- Bereitstellung einer eindeutigen und logischen Adressierung von Hosts
- Bestimmung möglichst optimaler Vermittlungspfade zwischen kommunizierenden Hosts (Routing)

Transportschicht Die Transportschicht realisiert eine Ende-zu-Ende-Kommunikation zwischen Prozessen auf unterschiedlichen Hosts.

- Flusskontrolle zur Verhinderung von Überlast beim Empfänger

- Staukontrolle zur Vermeidung von Überlastsituationen im Netz
- Multiplexing von Datenströmen verschiedener Anwendungen bzw. ihrer Instanzen
- Bereitstellung verbindungsloser bzw. -orientierter Transportmechanismen

5.5.0.5. Zusammenfassung: Schichten

Sitzungsschicht Die Sitzungsschicht erlaubt die Etablierung eines gemeinsamen Kommunikationszustandes, der mehrere Verbindungen auf der Transportschicht umfassen kann.

- Stellt Synchronisationspunkte für die Kommunikation zur Verfügung
- Ermöglicht damit das Suspendieren und Wiederaufnehmen von Kommunikationen
- Realisiert einen Koordinationsmechanismus für Kommunikationspartner

Darstellungsschicht Die Darstellungsschicht ermöglicht die Bereitstellung eines abstrakten Formats zur Repräsentation der übertragenen Daten.

- Abstraktion von anwendungsspezifischer Syntax
- Bereitstellung von Datenkompression
- Ver- und Entschlüsselung der Kommunikationsdaten
- Umkodierung der Kommunikationsdaten

5.5.0.6. Zusammenfassung: Schichten

Anwendungsschicht Die Anwendungsschicht beinhaltet alle Protokolle, welche direkt mit Anwendungen interagieren und deren Datentransport realisieren.

- Stellen einen Dienst für den User zur Verfügung
- Besitzen kein gemeinsames Dienste-Interface, da ihr Einsatzzweck sehr unterschiedlich ist

Die durch das ISO/OSI Modell realisierten Abstraktionen ermöglichen es Prozessen auf verschiedenen Systemen, transparent über ein Netz zu kommunizieren, ohne sich mit dem eigentlichen Übermittlungsvorgang auseinandersetzen zu müssen. Die Schichtarchitektur selbst erlaubt, dass die Technologien einzelner Segmente ausgetauscht werden können, ohne dass Änderungen am restlichen Stack vorgenommen werden müssen (z. B. kabelgebundene Kommunikation vs. WLAN). Zusätzlich ist es möglich, Adapter zur Verfügung zu stellen, welche die gleichzeitige Nutzung verschiedener Technologien während ein und demselben Datenaustausch ermöglichen (z. B. WLAN Access Point).

6. Netzsicherheit

6.1. Motivation

Warum ist Sicherheit in Netzen von Bedeutung?

- Das ISO/OSI-Modell stellt eine Netzarchitektur zur Verfügung, welche Systemen erlaubt, (zuverlässig) miteinander zu kommunizieren.
- Typischerweise findet diese Kommunikation über viele verschiedene Zwischensysteme (Router, Proxies, Mailserver) statt.
- Da der Austausch im Standardfall im Klartext durchgeführt wird, sind alle zwischengeschalteten Systeme in der Lage, die Daten mitzulesen und zu speichern.
- Bisher eingesetzten Mechanismen zur Integritätswahrung dienen lediglich eine Absicherung gegen (zufällige) Übertragungsfehler.
- Eine „sichere“ Kommunikation ist so nicht möglich.
- Zudem sind mittlerweile Netze aller Art Ziele von Angriffen, deren Techniken immer ausgefeilter werden.

Das Problemfeld vertraulicher und geschützter Kommunikation wird also im ISO/OSI-Modell nicht behandelt. Aspekte, welche die Sicherheit der Kommunikation betreffen, sind in diesem Modell nicht verankert. → weitere Strategien zur Erreichung dieser Ziele sind notwendig
Das beschriebene Referenzmodell stellt lediglich die grundlegende Funktionalität zur Kommunikation dar. Auf den verschiedenen Schichten kann es daher zu Angriffen kommen. Einige sollen im Folgenden kurz genannt werden.
Spoofing-Angriffe haben das Ziel die Adresse des Absenders zu fälschen, um somit die Identität des Angreifers zu verschleiern. Solche Angriffe können auf unterschiedlichen Layern durchgeführt werden, z.B. IP-Spoofing oder MAC-Spoofing. Eng damit verbunden ist hierbei auch ARP-Request-Poisoning, bei welchem gefälschte ARP-Pakete verschickt werden.
Weitere Angriffe aus dem Netzwerkbereich sind diverse Arten des **DoS (Denial of Service)**. Durch die Ausnutzung der Broadcast-Funktionalität in Direktverbindungsnetzen und des ICMP Protokolls können hier die Opfer überlastet werden. Der Angreifer sendet hierbei einen Ping an die Broadcastadresse und trägt als Absender die Adresse des Opfers ein. Alle Antworten (Echo Reply) werden dann an das Opfer versandt.
Auch durch das Propagieren falscher Routing-Informationen können Angreifer erheblichen Schaden anrichten, wenn beispielsweise durch die Angabe besonders guter Metriken, der Netzverkehr über den Angreiferknoten geleitet wird. Dieser kann dann zum einen den Verkehr beobachten und auch selektiv weiterleiten. Im schlimmsten Fall erzeugt er ein *Schwarzes Loch* und leitet gar keinen Verkehr weiter.
Weitere Schwachstellen und Angriffe seien hier nur vermerkt: Ping-of-Death, DNS-Cache Poisoning, TCP-Sequenznummern manipulation.

6.2. Grundlegende Begriffe

Um ein besseres Verständnis für das Thema Netzsicherheit zu erlangen, ist es notwendig, einige Grundbegriffe zu definieren.

6.2.1. Grundbegriffe

- Der Begriff Sicherheit umfasst eine Sammlung verschiedener Sicherheitsbegriffe, welche sich auf unterschiedliche zu schützende Werte (assets) beziehen.

Definition (Sicherheit):

Unter dem Begriff Betriebssicherheit (safety) versteht man hauptsächlich die technische Sicherheit, d. h. das System funktioniert unter normalen Umständen wie es soll. Insbesondere darf es keine Gefahrenquelle für Benutzer darstellen.
Informationssicherheit (security) hingegen besteht, wenn Informationen weder unautorisiert gewonnen noch verändert werden können.

Safety wird auch als funktionale Sicherheit bezeichnet und beschreibt den Schutz der Menschen und der Umwelt, d.h. das System selbst darf keinen Schaden anrichten. Security ist der Schutz des Systems selbst, sodass es keinen Schaden nimmt.

- Zudem ist der Begriff Datensicherheit (protection) zentral. Dieser behandelt den Schutz vor unautorisierter Manipulation von **Systemressourcen** sowie Maßnahmen zum Schutz vor Datenverlusten (z. B. durch Backups).Hier steht folgende Frage im Mittelpunkt: Wie werden die Daten gesichert?
- Der Begriff Datenschutz (privacy) ist im deutschen Bundesdatenschutzgesetz (BDSG) festgelegt und beschreibt die Selbstbestimmung natürlicher Personen über die Weitergabe der eigenen personenbezogenen Daten. Hier stehen folgende Fragen im Mittelpunkt: Wie werden Daten ordnungsgemäß erhoben und ordnungsgemäß verwendet? (z.B. Zweckbindungsgrundsatz)

Definition (Verlässlichkeit):

Ein System erfüllt die Eigenschaft der Verlässlichkeit (dependability), wenn es betriebssicher ist und die Funktionalitäten zuverlässig (Reliability) erfüllt. Hierzu können fehlertolerante Systeme eingesetzt werden.

6.2.2. Schutzziele

- Sicherheit in IT-Systemen im Allgemeinen fasst sowohl Informationen als auch die sie repräsentierenden Daten als zu schützende Werte auf.
- Um diese Werte schützen zu können, werden eine Reihe verschiedener Schutzziele definiert.

Definition (Authentizität):

Unter dem Begriff der Authentizität (authenticity) wird die Echtheit eines Objekts bzw. Subjekts verstanden. Die Echtheit kann mittels einer eindeutigen Identität oder charakteristischen Eigenschaft überprüfbar sein.

Ablauf eines Authentizitätsnachweises:

- Soll die Echtheit von A gegenüber B nachgewiesen werden, muss zuerst A einen vermeintlichen Nachweis der eigenen Authentizität erbringen (Authentisierung).
- Danach verifiziert B den dargebotenen Nachweis (Authentifikation) und schließt daraus auf die Authentizität von A.

Welche Verfahren sind hier möglich?

- Authentisierung durch Wissen, z. B. Passwörter, PIN Probleme: Diese können weitergegeben, oder vergessen werden.
- Authentisierung durch Besitz, z. B. Smartcards Probleme: Diese können allerdings verloren gehen oder gestohlen werden.
- Authentisierung durch biometrische Merkmale, z. B. Fingerabdruck, Iris Probleme: In diesem Fall sind spezielle Vorrichtungen zur Erhebung, d. h. zum Abscannen, notwendig, welche u. U. teuer sein können.

Eine Kombination der genannten Verfahren ist möglich (multi-factor authentication)

Definition (Datenintegrität):

Datenintegrität (integrity) beschreibt die Unveränderbarkeit der zu schützenden Daten, d. h. die Daten können nicht unautorisiert und unbemerkt manipuliert werden.

A-Priori Mechanismen

- Mit Hilfe eines entsprechenden Rechtesystems können einzelne Befugnisse (z. B. Leserechte, Schreibrechte, ...) bestimmte Daten betreffend an Identitäten oder Gruppen gebunden werden. Man spricht hierbei von Zugriffskontrolle.
- Um die Weiterverarbeitung der gewonnenen Informationen zu beschreiben und zu kontrollieren, werden Mechanismen der Informationsflusskontrolle verwendet. Diese Problematik ist eng mit der Informationsvertraulichkeit verbunden.

A-Posteriori Mechanismen

- Teilweise können Manipulationen nicht verhindert werden. Daher ist es notwendig, diese zu erkennen, um der Definition von Integrität gerecht zu werden.
- Hierfür können beispielsweise kryptografisch sichere Hashfunktionen verwendet werden (siehe S. 250ff).

Definition (Informationsvertraulichkeit):

Informationsvertraulichkeit (confidentiality) ist gegeben, wenn keine unautorisierte Gewinnung von Informationen möglich ist.

- Dies ist eng mit der Datenintegrität verbunden und setzt ebenfalls ein Rechtesystem voraus.

- Maßnahmen zur Bestimmung, Festlegung und Kontrolle zulässiger Datenflüsse sollen ausschließen, dass Informationen an unautorisierte Teilnehmer gelangen (Confinement Problem).
  Ein Beispiel für ein Sicherheitsmodell, welches Vertraulichkeit durchsetzt, ist das Bell-LaPadula Modell. Es basiert auf der Einteilung in verschiedene Schutzstufen, wobei gelten muss, dass aus niedrigeren Schutzstufen nicht in höhere gelesen werden kann und aus höheren Schutzstufen nicht in niedrigere geschrieben werden kann.

- Das sog. Interferenz Problem ist eng mit dem Confinement Problem verknüpft und beschreibt die Möglichkeit aus Einzelinformationen weitere Informationen ableiten zu können.

Diese drei Schutzziele spielen eine zentrale Rolle und werden meist unter dem Akronym CIA (Confidentiality – Integrity – Authenticity) zusammengefasst. Neben diesen wichtigen Schutzzielen gibt es noch eine Reihe weiterer Schutzziele, welche ebenfalls betrachtet werden müssen.

Definition (Verfügbarkeit):

Unter Verfügbarkeit (availability) versteht man, dass ein System seinen Dienst erbringt und seine Nutzer ihre Berechtigungen wahrnehmen können, ohne unautorisiert beeinträchtigt zu werden.

Definition (Verbindlichkeit):

Unter Verbindlichkeit (non-repudiation) versteht man die Fähigkeit eines Systems, eine Zuordnung zwischen Aktionen und Benutzern herzustellen, sodass Benutzer ihre durchgeführten Aktionen nicht abstreiten können.

Im Konflikt mit der Forderung der Verbindlichkeit stehen die Forderungen nach Privatsphäre. Es werden zunehemend Mechanismen wichtig, welche sich dem Aspekt der Privatsphäre widmen.

Definition (Anonymisierung, Pseudonymisierung):

Anonymisierung beschreibt das Verändern personenbezogener Daten, sodass ein Rückschluss auf die Identität nur mit unverhältnismäßig großem Aufwand möglich ist.
Eine schwächere Form der Anonymisierung wird durch die Pseudonymisierung erreicht, bei der die personenbezogenen Daten mit Hilfe einer Zuordnungsvorschrift von realen Personen getrennt werden.

Beispielsweise kann das auf Onion-Routing basierende Netzwerk TOR verwendet werden, wenn Verbindungen anonymisiert werden sollen. So kann anonym Web-Browsing oder IRC-Chat genutzt werden.

6.2.3. Angriffe, Bedrohungen und Risiko

- Eine Schwachstelle (weakness) bezeichnet eine Schwäche im System, z. B. unsichere Kommunikationskanäle.

- Eine Verwundbarkeit (vulnerability) ist eine Schwachstelle, über welche ein Angreifer die Sicherheitsmechanismen des Systems umgehen kann, z. B. Heartbleed.
- Eine Bedrohung (threat) ergibt sich dann, wenn Schwachstellen oder Verwundbarkeiten so ausgenutzt werden können, dass die Schutzziele gefährdet sind.
- Das Risiko einer Bedrohung ist die Eintrittswahrscheinlichkeit des Ereignisses, welches den Schaden verursacht, kombiniert mit der potentiellen Höhe des entstehenden Schadens.

Mit Hilfe der sog. CVEs (Common Vulnerabilites and Exposures) wird versucht Vulnerabilities zu sammeln und einheitlich zu beschreiben. Ein typisches Beispiel einer Vulnerability ist ein Pufferüberlauf (Buffer Overflow), welcher durch ungenügende Eingabeprüfungen in der Implementierung entstehen kann. Eine Bedrohung ist dann die Gefahr, dass ein solcher Buffer Overflow ausgenutzt werden kann um eine Priviledge Escalation zu erreichen (ein Nutzer erhält mehr Rechte, als zugewiesen, typischerweise Root-Access).
Als Angriff (attack) versteht man nun den (versuchten) unautorisierten Zugriff auf ein System. Man kann zwischen aktiven und passiven Angriffen unterscheiden.

Passiv	**Aktiv**
unautorisierte Informationsgewinnung	unautorisierte Modifikation
→ Verlust der Vertraulichkeit	→ Verlust der Integrität oder Verfügbarkeit

Beispiele für passive Angriffe sind etwa das Abhören der Kommunikationsleitung (eavesdropping) oder von Passwörtern (sniffing). Aktive Angriffe im Netzbereich sind vor allem das Verändern, Entfernen, Wiedereinfügen (Replay) von Nachrichten.
Zwei spezielle Klassen von aktiven Angriffen sind zudem die sog. Maskierung (Spoofing) und Denial-of-Service-Angriffe.
Ersteres richtet sich gegen die Verbindlichkeit. DoS-Angriffe richten sich gegen die Verfügbarkeit.

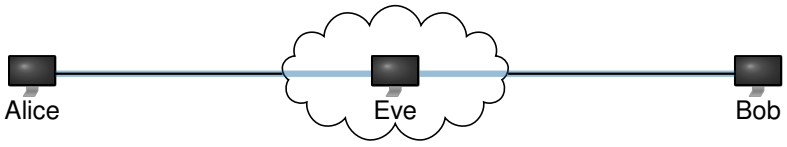
6.3. Kryptografie

Um ein grundlegendes Verständnis für Verschlüsselungstechniken und weitere Krypthografische Verfahren zu schaffen, werden zu nächst einige wichtige Begriffe eingeführt.

6.3.1. Kryptografische Grundlagen

Die bisherigen Probleme und Schutzziele lassen sich durch folgende **Problemstellung** auf die Netzwerkdomäne beziehen.

- Alice und Bob wollen miteinander kommunizieren.



- Alice hat keinen Einfluss darauf, wie Datenpakete durch das Internet geleitet werden, wer sie mitlesen oder sogar modifizieren kann.

- Alice kann sich (trotz korrekter Adressierung von Bob) nicht einmal sicher sein, überhaupt mit Bob zu kommunizieren:
  - Der Angreifer Eve könnte sich sowohl Alice als auch Bob gegenüber als der jeweils andere Kommunikationspartner ausgeben.
  - Einzige Voraussetzung dafür ist, dass Eve die Pakete von Alice und Bob „abfangen“ kann.
  - Befindet sich Eve im lokalen Netz von Alice oder Bob, ist das sehr einfach: ARP-Spoofing.

Idee: Alice und Bob authentifizieren sich gegenseitig und verschlüsseln ihre Daten, sodass

- sichergestellt ist, dass Alice und Bob auch wirklich diejenigen sind, die sie vorgeben zu sein, und
- nur der jeweils andere Kommunikationspartner in der Lage ist, die Daten wieder zu entschlüsseln.

Ziele kryptografischer Verfahren Mittels kryptographischer Verfahren werden im Speziellen folgende bereits beschriebenen Schutzziele verfolgt:

- Integrität
  Bob will sich sicher sein, dass die Daten von Alice auf dem Weg nicht verändert wurden.
- Authentizität
  Bob will sich sicher sein, dass die Daten auch wirklich von Alice stammen und nicht von jemandem, der sich für Alice ausgibt.
- Vertraulichkeit
  Es soll verhindert werden, dass unbefugte Dritte Nachrichten zwischen Alice und Bob mitlesen können.
- Verbindlichkeit / Nichtabstreitbarkeit
  Dem Sender einer Nachricht soll es nicht möglich sein, zu bestreiten, dass er der Urheber einer bestimmten Nachricht ist.

Diese Ziele werden i. d. R. nur durch das (komplizierte) Zusammenspiel aus

- Sitzungsprotokollen
- Schlüsselaustauschverfahren bzw. -protokollen,
- Verschlüsselungsalgorithmen und
- Hashverfahren

erreicht.

6.3.1.1. Grundlegende Terminologie

Definition (Kryptographie, Kryptoanalyse, Kryptologie):

Unter dem Begriff Kryptographie versteht man Methoden zur Absicherung von Nachrichten mit Hilfe von Verschlüsselung. Mit dem Begriff Kryptoanalyse wird das Vorgehen beschrieben, diese Verschlüsselung zu brechen und die Klartext zu reproduzieren. Kryptologie verbindet Kryptographie und Kryptoanalyse.

Ein weiterer Teilaspekt der Kryptologie bildet die Steganografie, bei welcher versucht wird, Informationen zu verstecken (z.B. in Bild- oder Audiodateien).

Die Ziele der Kryptoanalyse sind vielfältig. Im besten Fall ist es möglich den verwendeten Schlüssel zu bestimmen. Ein schwächeres Ergebnis ist die globale Deduktion, welche einen alternativen Entschlüsselungsalgorithmus liefert. Noch schwächer ist die lokale Deduktion, bei welcher der Klartext zu bestehendem Chiffretext bestimmt werden kann. Am schwächsten ist die Informationsdeduktion, welche lediglich eingeschränkte Informationen zum Klartext oder Schlüssel liefern kann.
Grundlage der Kryptoanalyse sind verschiedene Angriffsmöglichkeiten.

- Bei Ciphertext-Only steht nur eine bestimmte Menge an Chiffretexten zur Verfügung.
- Bei Known-Plaintext ist zu einer Menge an Chiffretexten eine Menge korrespondierender Klartexte vorhanden.
- Bei Choosen-Plaintext können beliebige Klartexte vorgegeben und die entsprechenden Chiffretexte erzeugt werden.
- Bei Choosen-Cyphertext können zu beliebigen vorgegebenen Chiffretexten, die entsprechenden Klartexte erzeugt werden.

Ein **Kryptosystem KS** kann dargestellt werden als ein Sechstupel

$$\boldsymbol{KS = (\mathcal{M}, \mathcal{C}, EK, DK, E, D)}$$

mit

- $\mathcal{M}$: einer nicht-leeren Menge von **Klartexten** $\mathcal{M} \subseteq A_1^*$, wobei A_1^* die Menge aller Worte über dem Alphabet A_1 beschreibt
- $\mathcal{C}$: einer nicht-leeren Menge von **Chiffretexten** $\mathcal{C} \subseteq A_2^*$
- Die **Alphabete** sind dabei endliche Mengen von Zeichen deren Mächtigkeit mit $|\mathcal{A}| = n$ beschrieben wird. Die Alphabete A_1^* und A_2^* können, müssen aber nicht, gleich sein.
- EK: einer nicht-leeren Menge von **Verschlüsselungschlüsseln**
- DK: einer nicht-leeren Menge von **Entschlüsselungsschlüsseln** sowie einer Bijektion $f : EK \to DK$, sodass $\exists K_E \in EK : f(K_E) = K_D$
- E: einem linkstotalen und injektiven **Verschlüsselungsverfahren**
- D: einem korrespondierenden **Entschlüsselungsverfahren**

Definition: injektiv

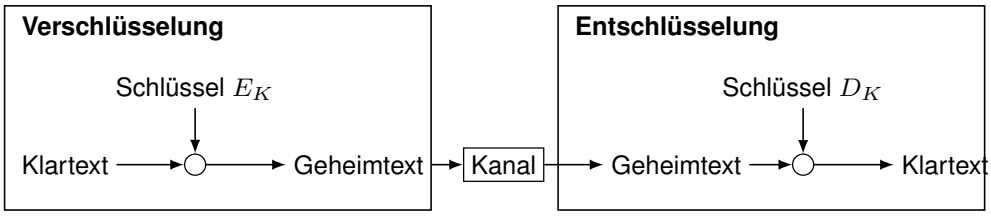
$$f\colon D \to B\colon \forall x_1, x_2 \in D\colon x_1 \neq x_2 \Rightarrow f(x_1) \neq f(x_2)$$

Definition: linkstotal

$$f\colon D \to B\colon \forall x \in D \exists y \in B\colon f(x) = y$$

6.3.1.2. Verschlüsselungsverfahren

- **Verschlüsselung (Encryption)** beschreibt die Abbildung von $M \in \mathcal{M}$ auf $C \in \mathcal{C}$: $E(M) = C$.
- **Entschlüsselung (Decryption)** beschreibt die Umkehrfunktion: $D(C) = M$
- Daher gilt: $D(E(M)) = M$



Für Verschlüsselungsverfahren soll im Allgemeinen gelten:

Kerkhoffs-Prinzip:

Die Sicherheit eines Verschlüsselungsverfahrens darf nur von der Geheimhaltung des Schlüssels abhängen, also auch nicht von der Geheimhaltung des Verschlüsselungsalgorithmus.

Das Kerkhoffs-Prinzip wurde bereits 1883 aufgestellt und richtet sich prinzipiell gegen Security by Obscurity. Ein vollständiges Verfahren unter Verschluss zu halten gestaltet sich im Allgemeinen schwierig, da beispielsweise Beteiligte den Arbeitsplatz wechseln könnten. Über dies bekommt die Community der Sicherheitsexperten so keine Möglichkeit ein Verfahren zu prüfen. Ein Beispiel für die Nicht-Einhaltung des Prinzips sind die Stromchiffren A5 1 und A5 2, welche bei GSM zum Einsatz kamen und heute in Echtzeit gebrochen werden können. Generell kann jede Verschlüsselung mittels Brute-Force-Angriffen, d.h. stupidem Ausprobieren aller Schlüsselkombinationen, gebrochen werden. Daher ist die Schlüssellänge und der somit entstehende Schlüsselraum von zentraler Bedeutung. Dieser sollte möglichst groß gewählt werden, sodass der Angreifer praktisch keine Möglichkeit hat, alle Kombinationen zu testen.

Klassifizierung von Verschlüsselungsverfahren Symmetrische Verfahren

- Sender und Empfänger einigen sich auf ein gemeinsames Geheimnis (engl. Shared Secret, ugs. „Passwort“ oder „Schlüssel“).
- Mittels dieses Schlüssels können Daten verschlüsselt und auch wieder entschlüsselt werden (daher „symmetrische“ Verfahren).
- Das bedeutet: $E_K(M) = C$ und $D_K(C) = M$, woraus folgt: $D_K(E_K(M)) = M$
- Man beachte, dass für die Ent- sowie Verschlüsselung dasselbe K verwendet wird.

Asymmetrische Verfahren

- Alice und Bob besitzen jeweils zwei Schlüssel: Einen Public Key und einen Private Key.

- Der öffentliche Schlüssel ist prinzipiell jedem zugänglich, der private Schlüssel wird geheim gehalten.
- Alice verschlüsselt eine Nachricht mit dem öffentlichen Schlüssel von Bob.
- Zur Entschlüsselung ist der passende private Schlüssel notwendig, den nur Bob besitzt.
- Das bedeutet: $E_{K_{pub}}(M) = C$ und $D_{K_{priv}}(C) = M$, woraus folgt: $D_{K_{priv}}(E_{K_{pub}}(M)) = M$
- Man beachte, dass hier K_{priv} der private und K_{pub} der öffentliche, d.h. zwei verschiedene Schlüssel sind.
- *Praktischer Einsatz:* Dieses Prinzip steht hinter den RSA-Schlüsseln, die Sie zum Zugriff auf Ihre GRNVS-VMs benötigen.

Eine Möglichkeit die öffentlichen Schlüssel zugänglich zu machen, sind sogenannte Key-Server auf welchen die öffentlichen Schlüssel geladen werden. Der private verbleibt beim Besitzer. Es besteht generell keine Garantie, dass der auf einem Key-Server hinterlegte öffentliche Schlüssel dem angegebenen Besitzer gehört.

Stromchiffren

- Bei Stromchiffren wird ein Zeichen nach dem anderen verschlüsselt.
- Im Folgenden wird der RC4 (Rivest Cipher 4) Algorithmus als Vertreter der Stromchiffren sowie der symmetrischen Verschlüsselungsverfahren betrachtet.

Blockchiffren

- Bei Blockchiffren wird die zu verschlüsselnde Nachricht in einzelne Blöcke (z. B. 64 Bit) zerlegt.
- Anschließend wird ein Block nach dem anderen verschlüsselt.
- Zwei wichtige Vertreter der Blockchiffre sind der DES[1]- und der AES-Algorithmus.
- Blockchiffren können in verschiedenen Modi verwendet werden.
- Im Folgenden betrachten wir zwei einfache Vertreter zur Veranschaulichung: ECB (Electronic Codebook Mode) und CBC (Cipherblock Chaining Mode).

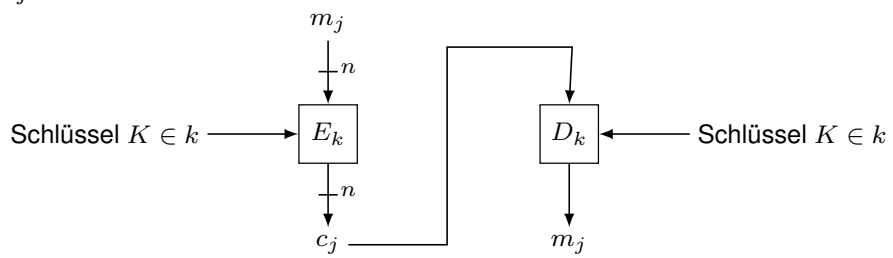
DES steht hierbei für Data Encryption Standard und weist eine Schlüssellänge von 56 Bit auf. AES bedeutet Advanced Encryption Standard.
Eine Erweiterung des DES ist der sog. Triple DES, welcher den DES mit drei verschiedenen Schlüsseln anwendet, indem der zu verschlüsselnde Text zunächst mit DES verschlüsselt, dann wieder (mit einem anderen Schlüssel) entschlüsselt und zuletzt nochmals verschlüsselt wird. Bei allen Schritten werden verschiedenen Schlüssel benutzt. Durch spezielle Angriffstechniken wird die effektive Schlüssellänge von theoretisch maximalen 168 Bit auf nur noch 112 Bit verkürzt. Werden für die beiden Verschlüsselungsschritte die gleichen Schlüssel verwendet, verbleiben lediglich 80 Bit effektive Schlüssellänge.

[1] DES gilt heute als unsicher und sollte daher nicht mehr verwendet werden.

6.3.2. Block- und Stromchiffren

6.3.2.1. Blockchiffren

Blockchiffren – ECB Mode *Hinweis:* Auf der folgenden Grafik ist sowohl die Ver- als auch die Entschlüsselung dargestellt. Die c_j bezeichnen die jeweiligen Chiffretextblöcke, alle m_j beziehen sich auf den Klartext.



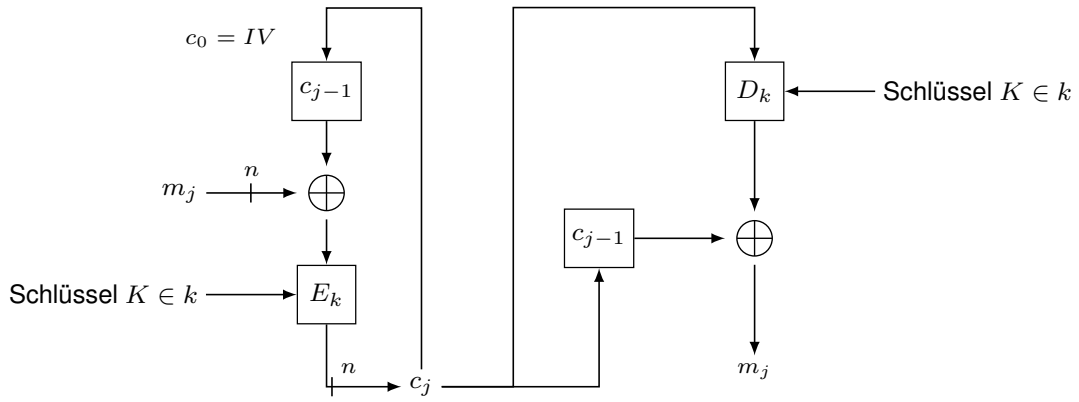
- Beliebig lange Klartexte werden in Blöcke der Länge n aufgeteilt. Geht die Zerlegung nicht auf, muss der Klartext gepaddet werden.
- Hierbei ist zu beachten, dass mit Zufallsbits gepaddet wird, da dem Angreifer sonst die Möglichkeit auf einen Known-Plaintext gegeben wird.
- Jeder Block wird eigenständig verschlüsselt und wieder entschlüsselt.
- Die Konkatenation der einzelnen entschlüsselten Blöcke ergibt die ursprüngliche Nachricht.
- Dieses Verfahren ist für symmetrische wie auch asymmetrische Verschlüsselung möglich.

Beispiel: $m = 110100111001$

- Es ergeben sich mit $n = 4$ folgende Blöcke: $m_1 = 1101$, $m_2 = 0011$ und $m_3 = 1001$
- Mittels Verschlüsselung durch Permutation $\pi = \begin{pmatrix} 1 & 2 & 3 & 4 \\ 4 & 3 & 2 & 1 \end{pmatrix}$
- ergibt sich der folgende Chiffretext: $c = 1011\ 1100\ 1001$

Eine Permutation (Transpositionschiffre) verändert keine Zeichen, sondern lediglich deren Positionen. Anstelle der im Beispiel verwendeten Permutation könnte auch eine Substitution verwendet werden, welche nicht die Positionen der Zeichen, sondern die Zeichen selbst verändert. Hierbei kann zwischen mono- und polyalphabetischen Substitutionen unterschieden werden. Letztere können verwendet werden, um die Auftrittswahrscheinlichkeiten der Zeichen im Klartext zu verschleiern (homophone Chiffren).

Blockchiffren – CBC Mode *Hinweis:* Auf der folgenden Grafik ist sowohl die Ver- als auch die Entschlüsselung dargestellt. Bezeichnungen sind analog zur vorherigen Abbildung.



- Es wird ein fester Initialisierungsvektor (IV) der Blocklänge n und als Verknüpfung *XOR* verwendet .
- Der Klartext wird auch hier in feste Blöcke der Länge n aufgeteilt.

Blockchiffren – CBC Mode: Anwendung und Beispiel Für die Verschlüsselung der Klartextblöcke $m_1, ..., m_x$ der Länge n mit Schlüssel k wird wie folgt vorgegangen:

- $c_0 = IV$ und $c_j = E_k(c_{j-1} \oplus m_j)$ für alle $1 \leq j \leq x$
- Ergebnis: Chiffretextblöcke $c_1, ..., c_x$

Für die Entschlüsselung der Chiffretextblöcke $c_1, ..., c_x$ der Länge n mit Schlüssel k wird wie folgt vorgegangen:

- $c_0 = IV$ und $m_j = c_{j-1} \oplus D_k(c_j)$ für alle $1 \leq j \leq x$
- Ergebnis: Klartextblöcke $m_1, ..., m_x$
- Die Schlüssel für Ver- und Entschlüsselung müssen nicht notwendigerweise gleich, aber korrespondierend sein.

Im Folgenden betrachten wir erneut das bekannte Beispiel: $m = 110100111001$

- Es ergeben sich mit $n = 4$ folgende Blöcke: $m_1 = 1101$, $m_2 = 0011$ und $m_3 = 1001$
- Mittels Verschlüsselung durch Permutation $\pi = \begin{pmatrix} 1 & 2 & 3 & 4 \\ 4 & 3 & 2 & 1 \end{pmatrix}$
- IV sei gegeben mit 1011
- ergibt sich der folgende Chiffretext: $c = 0110\ 1010\ 1100$

Bewertung der beiden Modi *Nachteile und Probleme im ECB-Mode*

- Gleiche Klartextblöcke werden in gleiche Chiffretextblöcke umgewandelt, daher übertragen sich Regelmäßigkeiten und Muster des Klartextes in den Chiffretext.
- Diese Informationen können die Kryptoanalyse unterstützen.
- Der Angreifer kann zudem unbemerkt Chiffretext einfügen, der mit demselben Schlüssel verschlüsselt wurde.

- Eine Änderung der Blockreihenfolge bleibt zudem ebenfalls unbemerkt.

Der ECB-Mode sollte daher nicht für längere Nachrichten benutzt werden.

- Gleiche Texte werden bei geändertem Initialisierungsvektor unterschiedlich verschlüsselt.
- Gleiche Klartextblöcke werden unterschiedlich (kontextabhängig) verschlüsselt.
- Änderungen des Chiffretextes oder seiner Reihenfolge werden erkannt.

Nachteile und Probleme im CBC-Mode

- Übertragungsfehler in einem Block werden in den nächsten Block propagiert (jedoch nicht weiter).
- Benutzen Sender und Empfänger nicht denselben Initialisierungsvektor, kann der erste Block nicht entschlüsselt werden.

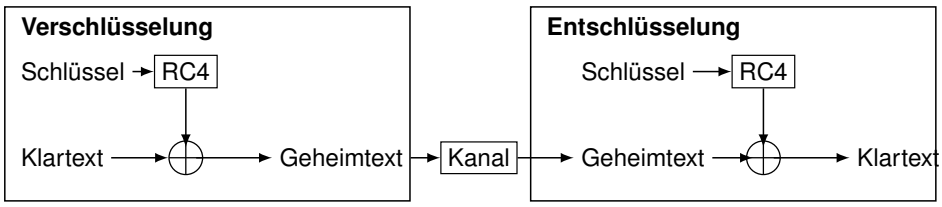
6.3.2.2. Stromchiffren: RC4

Verschlüsselung:

- Der Sender erzeugt mit Hilfe eines gemeinsamen Geheimnisses einen Schlüsselstrom (engl. cipher stream).
- Da der Bitstrom pseudozufällig ist, kann er mit Hilfe des gemeinsamen Geheimnisses von beiden Kommunikationspartnern reproduziert werden.
- Der Schlüsselstrom wird mit der zu verschlüsselnden Nachricht (Klartext, engl. plain text) bitweise XOR verknüpft.
- Der resultierende Datenstrom wird als Geheimtext (engl. cipher text) bezeichnet.

Entschlüsselung:

- Der Empfänger kann mit Hilfe des gemeinsamen Geheimnisses denselben Schlüsselstrom erzeugen.
- XOR-Verknüpfung aus Schlüsselstrom und Geheimtext ergibt wieder den Klartext.



Erzeugung des Schlüsselstroms (KSA, key-scheduling algorithm):

- RC4 initialisiert in Abhängigkeit des Schlüssels eine 256 Byte lange sog. Substitutions-Box (S-Box) mit den Zahlen 0 bis 255.

- Im Anschluss werden in jedem Schritt zwei Zahlen aus der S-Box permutiert und die Summe der beiden permutierten Zahlen modulo 256 als Index zu einer neuen Zahl in der S-Box verwendet, welche als „Zufallszahl“ zurückgegeben wird.[2]

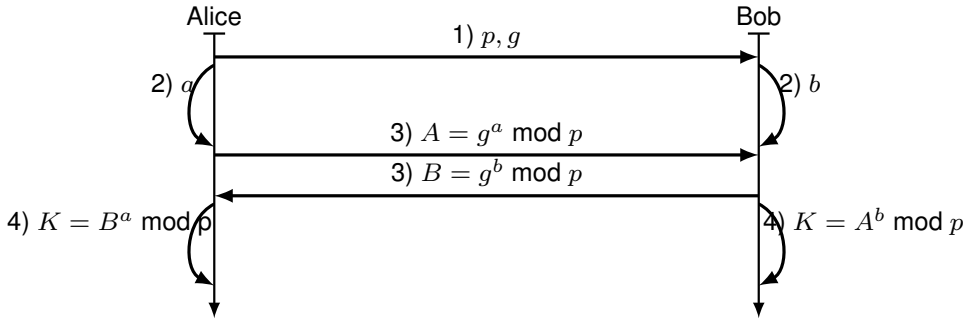
Bei RC4 ist zu beachten:

- Es dürfen niemals zwei Nachrichten mit demselben Schlüsselstrom verschlüsselt werden, da andernfalls ein Angreifer, dem eine Nachricht im Klartext bekannt ist, auch die andere Nachricht im Klartext berechnen kann.
- Aufgrund der einfachen Funktionsweise lassen die ersten Bytes des Schlüsselstroms Rückschlüsse auf den Schlüssel selbst zu und sollten daher verworfen werden.
- Wie jeder Generator für pseudozufällige Zahlen hat RC4 eine begrenzte Periodendauer, d. h. irgendwann wiederholt sich der Schlüsselstrom.
- Bei RC4 wurden bestimmte Korrelationen zwischen Schlüssel und Schlüsselstrom gefunden, so dass RC4 heute als nicht mehr sicher angesehen wird.

Verschlüsselungsprotokolle (Cipher Suites), die RC4 nutzen:

- WEP[3] (Wired Equivalent Privacy)
- WPA-TKIP (WiFi Protected Access Temporal Key Integrity Protocol)
- Als optionaler Algorithmus in IPSec, SSL/TLS, SSH, Kerberos

Wie tauschen Sender und Empfänger die gemeinsamen Schlüssel aus? Eine Möglichkeit besteht in der Nutzung des Diffie-Hellman-Verfahrens:



1. Alice und Bob einigen sich zunächst auf eine Primzahl p und eine sog. primitive Kongruenzwurzel[4] g mod p.
2. Beide erzeugen unabhängig voneinander jeweils eine Zufallszahl a und b.
3. Nun werden die Zahlenwerte $B = g^b \bmod p$ bzw. $A = g^a \bmod p$ ausgetauscht. Die Zufallszahlen a und b bleiben aber geheim.
4. Alice und Bob können nun jeweils $A^b \bmod p = B^a \bmod p = K$ berechnen.

[2] Diese Darstellungsweise ist etwas vereinfachend.
[3] Die Schwäche liegt hier nicht bei RC4 selbst sondern bei der Art, wie RC4 genutzt wird.
[4] g ist eine primitive Kongruenzwurzel zur Primzahl p, wenn $g^i \bmod p$ für $i = \{1, 2, \ldots, p-1\}$ alle Zahlen zwischen 0 und einschließlich $p-1$ erzeugt. Werte für p und g finden sich u.a. in RFC5114.

Funktioniert das denn? Zu zeigen: $K \stackrel{!}{=} A^b \bmod p \stackrel{!}{=} B^a \bmod p$

$$\begin{aligned} A^b \bmod p &= (g^a)^b \bmod p \\ &= \left(g^b\right)^a \bmod p \\ &= B^a \bmod p \end{aligned}$$

Ist das Verfahren sicher? Gegenfrage: Was ist sicher?

- RC4 und das Diffie-Hellman-Verfahren wie hier vorgestellt sind anfällig gegenüber Man-in-the-Middle-Angriffe:
  - Schafft es Eve, während des Verbindungsaufbaus sowohl die Nachrichten von Alice als auch von Bob abzufangen, kann sie sich für den jeweils anderen Kommunikationspartner ausgeben und unterschiedliche Schlüssel mit Alice und Bob aushandeln.
  - Nach dem Schlüsselaustausch muss Eve weiterhin in der Lage sein, alle Nachrichten von Alice bzw. Bob abzufangen.
  - Diese können dann entschlüsselt, gelesen und modifiziert und anschließend neu verschlüsselt und weitergeleitet werden.
- Aus diesem Grund ist keines der Kriterien Integrität, Authentizität, Vertraulichkeit und Verbindlichkeit wirklich erfüllt.

⇒ RC4 und das Diffie-Hellman-Verfahren alleine sind ein Anfang. Für eine sichere Kommunikation wird aber ein komplettes Protokoll benötigt.

6.3.3. Integritätsschutz und Digitale Signaturen

6.3.3.1. Hashfunktionen

Mittels kryptografisch sicheren Hashfunktionen soll die Integrität der Nachrichten sichergestellt werden. Man kann zwischen starken und schwachen Hashfunktionen unterscheiden. Eine injektive Funktion $H : A^* \to B^k$ über die Alphabete A und B ist eine schwache Hashfunktion, wenn gilt:

- H ist eine Einweg-Funktion, d.h. die Umkehrfunktion ist nicht effizient berechenbar
- Der Hashwert $H(M) = h$ mit $|h| = k$ ist für gegebene M effizient berechenbar
- Es ist praktisch unmöglich, ein M' zu bestimmen, für das bei gegebenem $h = H(M)$ gilt: $h = H(M')$ mit $M \neq M'$
- Hierbei geht es darum, für einen gegebenen Hash eine Kollision zu finden.

Eine starke Hashfunktion ist eine schwache Hasfunktion, für welche zusätzlich gilt:

- Es ist praktisch unmöglich, ein Paar von Nachrichten M und M' mit $M \neq M'$ zu bestimmen, für das gilt: $H(M) = H(M')$
- In diesem Fall soll es praktisch unmöglich sein, irgendeine Kollision zu finden.

Ein Beispiel für eine Hashfunktion ist die Modulo-Operation. Beispiele für kryptografische Haschfunktionen sind der MD4 bzw. MD5 und SHA-1, welche jedoch alle als unsicher gelten. Andere Mitglieder der SHA Familie gelten noch als sicher.
Vorgehen bei der Nachrichtenüberprüfung mit Hinblick auf Integrität:

- Der Sender bestimmt zu Nachricht M den Hashwert $h = H(M)$ und sendet das Tupel (M, h)
- Der Empfänger bestimmt für die erhaltene Nachricht M' den zugehörigen Hashwert $H' = H(M')$
- Die Nachricht ist integer, wenn gilt: $h = h'$

6.3.3.2. Digitale Signaturen

- Bisher konnte Integritätsschutz der Nachricht sicher gestellt werden.
- Es ist aber weiterhin möglich, die Nachricht zu manipulieren und den Hash erneut zu berechnen.
- Im nächsten Schritt soll dies nun verhindert werden, indem auch der Sender der Nachricht mit dem Inhalt verbunden wird.

Definition (Digitale Signatur):

Um ein elektronisches Dokument zu unterschreiben, werden digitale Signaturen verwendet. Diese stellen die Verbindlichkeit bezüglich der signierenden Partei sicher und ermöglichen die Verifikation, dass ein Dokument auch wirklich von der signierenden Partei stammt.

Eigenschaften digitaler Signaturen:

- authentisch, d.h. die signierende Partei hat willentlich unterschrieben
- fälschungssicher, d.h. die signierende Partei hat unterschrieben und niemand anders
- nicht wiederverwendbar, d.h. die Signatur kann nicht auf ein anderes Dokument angewandt werden
- unveränderbar, d.h. nach dem Signieren kann die Unterschrift nicht unbemerkt geändert werden
- bindend, d.h. der Unterzeichner kann das Unterschreiben nicht abstreiten

6.3.3.3. Verschiedene Umsetzungsmöglichkeiten

Message Authentication Code

- Die Kommunikationspartner einigen sich auf einen geheimen Schlüssel K.
- Dieser wird zusätzlich in die Hashfunktion eingebracht.
- Das weitere Vorgehen ist dann analog zur Bestimmung der Integrität mittels Hashfunktionen: $\mathrm{MAC}(M', K) = mac' \stackrel{?}{=} mac = \mathrm{MAC}(M, K)$

Bei asymmetrischen Verfahren (z.B. RSA):

- Eine Nachricht m soll mittels einer digitalen Signatur signiert werden.
- Hierfür wird der private (geheime) Schlüssel K_{priv} verwendet: $E(K_{priv}, m) = sig_m$
- das signierte Dokument sig_m kann nun verschickt werden.
- Diese Signatur kann mit dem öffentlichen Schlüssel K_{pub} verifiziert werden: $m = D(sig_m, K_{pub})$
- Dieser öffentliche Schlüssel befindet sich öffentlich zugänglich, z.B. auf einem Key-Server.

6.4. Public Key Infrastrukturen (PKI)

6.4.1. Motivation

Bisher gelöste Probleme:

- Die Nachricht kann nicht von Unbefugten gelesen werden (Verschlüsselung)
- Die Integrität der Nachricht ist gesichert (Hashfunktionen)
- Der Sender der Nachricht kann verifiziert werden (Digitale Signatur mittels MAC)

Kann man wirklich sicher sein, dass Alice auch Alice ist?

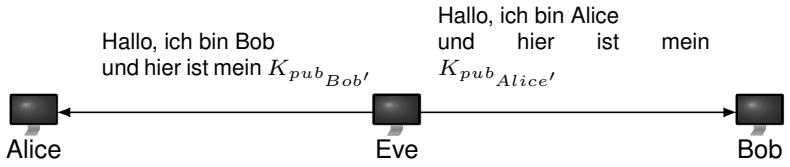
Definition (Man-in-the-Middle (MITM) Attack):

Unter dem Begriff Man-in-the-Middle (MITM) Attack versteht man einen Angriff auf Rechnernetze bei denen sich ein Angreifer logisch oder physikalisch zwischen die beiden Kommunikationsendpunkte bringt und somit den gesamten Netzverkehr abhören und manipulieren kann.

- Dieser Angriff untergräbt die bisherigen Sicherheitsmechanismen, da nicht sicher gestellt ist, dass die Absicherungen mit dem entsprechenden beabsichtigten Kommunikationspartner erfolgen.
- Im Folgenden soll der MITM Angriff kurz illustriert werden und entsprechende Absicherungen aufgezeigt werden.

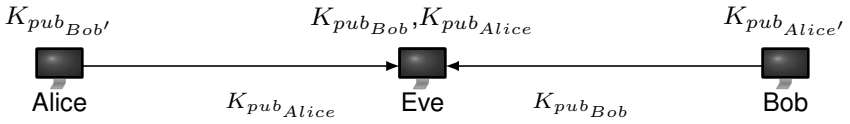
6.4.2. MITM - Angriff

- Alice möchte mit Bob kommunizieren und beide besitzen ein Schlüsselpaar bestehend aus einem öffentlichen und einem privaten Schlüssel.

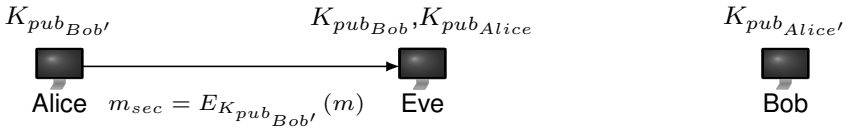


- Eve sendet ihren öffentlichen Schlüssel an Alice und Bob.

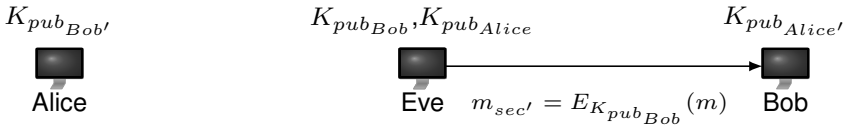
- Sowohl Alice als auch Bob besitzen nun die vermeintlichen Schlüssel des jeweils anderen.



- Alice und Bob können nun auch ihre öffentlichen Schlüssel austauschen, sodass auch Eve die beiden entsprechenden Schlüssel hat.



- Alice sendet nun eine geheime Nachricht m unter Zuhilfenahme von $K_{pub_{Bob'}}$ an Bob.



- Eve kann m_{sec} mit Hilfe ihre privaten Schlüssels entschlüsseln und durch Zuhilfenahme von Bobs öffentlichen Schlüssel unbemerkt weiterleiten.

- Bob kann die Nachricht mit Hilfe seines privaten Schlüssels entschlüsseln und alle Beteiligten (auch Eve) sind nun in Besitz der geheimen Nachricht.

Die Nutzung von Certification Authorities oder dem Web of Trust können hier helfen.

6.4.3. Zertifikate

Definition (Zertifikate):

Ein Zertifikat bindet einen öffentlichen Schlüssel an eine natürliche oder juristische Person.

- Die Struktur eines Zertifikats ist im *Standard X.509* festgelegt.
- Dieser Standard ist Teil eines größeren Frameworks, von allerdings nur noch dieser Teil von praktischer Relevanz ist.

Bezeichnung	Beschreibung
Versionsnummer	Angabe des verwendeten Zertifikatformats
Seriennummer	eindeutige ID
Signatur	benutzte Parameter und Algorithmen
Zertifikataussteller	Name der Instanz, welche das Zertifikat ausstellt
Gültigkeitsdauer	Zeitintervall
Benutzername	eindeutiger Name des Benutzers
Schlüsselinformationen	verwendeter Schlüssel des Benutzers inkl. Algorithmen

- Der Aussteller ist verantwortlich, dass die Seriennummer eindeutig bleibt.
- Um ein Zertifikat verifizieren zu können, müssen die verwendeten Hash- und Signierverfahren, sowie der öffentliche Schlüssel des Inhabers im Zertifikat enthalten sein.
- Das Zertifikat ist mit dem privaten Schlüssel des jeweiligen Ausstellers signiert.

Die erste Version dieses Standards (v1) erschien bereits 1988.
Seither gab es diverse Erweiterungen, wie beispielsweise ein *key usage - Feld*, welches den Zweck Schlüssels beschreibt (Verschlüsseln, Signieren, Zertifizieren). Weiterhin wurde mit der *private key usage periode* die Möglichkeit geschaffen, dem privaten Schlüssel eine andere Lebensdauer zuzuweisen, als dem Zertifikat selbst. Zudem kann mit Hilfe der *certification policies* beschrieben werden, unter welchen Bedingungen das Zertifikat ausgestellt wurde.

Beispiel - Root Certificate der Telekom

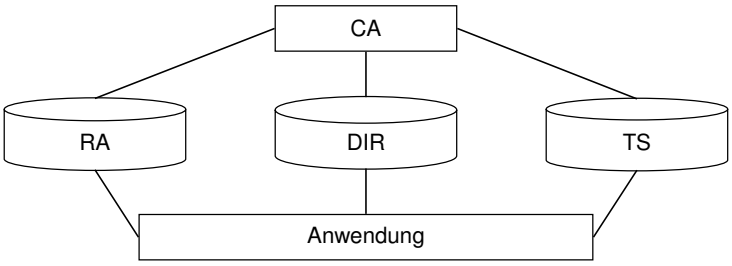
```
Certificate:

Data:
Version: 3 (0x2)
Serial Number: 38 (0x26)
Signature Algorithm: sha1WithRSAEncryption
Issuer: C=DE, O=Deutsche Telekom AG, OU=T-TeleSec Trust Center,
CN=Deutsche Telekom Root CA 2
Validity
Not Before: Jul  9 12:11:00 1999 GMT
```

```
Not After : Jul  9 23:59:00 2019 GMT
Subject: C=DE, O=Deutsche Telekom AG, OU=T-TeleSec Trust Center,
CN=Deutsche Telekom Root CA 2
Subject Public Key Info:
Public Key Algorithm: rsaEncryption
RSA Public Key: (2048 bit)
Modulus (2048 bit):
00:ab:0b:a3:35:e0:8b:29:14:b1:14:85:af:3c:10:
e4:39:6f:35:5d:4a:ae:dd:ea:61:8d:95:49:f4:6f:
...
bc:da:03:34:d5:8e:5b:01:f5:6a:07:b7:16:b6:6e:
4a:7f
Exponent: 65537 (0x10001)

X509v3 extensions:
X509v3 Subject Key Identifier:
31:C3:79:1B:BA:F5:53:D7:17:E0:89:7A:2D:17:6C:0A:B3:2B:9D:33
X509v3 Basic Constraints:
CA:TRUE, pathlen:5
X509v3 Key Usage: critical
Certificate Sign, CRL Sign

Signature Algorithm: sha1WithRSAEncryption
94:64:59:ad:39:64:e7:29:eb:13:fe:5a:c3:8b:13:57:c8:04:
24:f0:74:77:c0:60:e3:67:fb:e9:89:a6:83:bf:96:82:7c:6e:
...
57:99:94:0a:6d:ba:39:63:28:86:92:f3:18:84:d8:fb:d1:cf:
05:56:64:57

-----BEGIN CERTIFICATE-----
MIIDnzCCAoegAwIBAgIBJjANBgkqhkiG9w0BAQUFADBxMQswCQYDVQQGEwJERTEc
MBoGA1UEChMTRGV1dHNjaGUgVGVsZWtvbSBBRzEfMB0GA1UECxMWVC1UZWxlU2Vj
IFRydXN0IENlbnRlcjEjMCEGA1UEAxMaRGV1dHNjaGUgVGVsZWtvbSBSb290IENB
...
6iFhkOQxIY4OsfcvNUqFENrnijchvllj4PKFiDFT1FQUhXB59C4Gdyd1Lx+4ivn+
xbrYNuSD7Odlt79jWvNGr4GUN9RBjNYj1h7P9WgbRGOiWrqnNVmh5XAFmw4jV5mU
Cm26OWMohpLzGITY+9HPBVZkVw==
-----END CERTIFICATE-----
```

6.4.4. Zertifizierungsstellen

6.4.4.1. Zertifizierungsstellen

- Zertifizierungsinstanz (Certification Authority (CA)) bieten Dienste zur Ausstellung von Zertifikaten an.
- Das ausgestellte Zertifikat wird mit dem privaten Schlüssel der CA signiert.
- Die Gesamtheit der benötigten Komponenten wird in diesem Zusammenhang meist als Public-Key Infrastructure (PKI) bezeichnet.



- Die Registration Authority (RA) übernimmt die Verknüpfung zwischen öffentlichem Schlüssel und Indentitäten.
- Der Verzeichnisdienst (DIR), z. B. in Form eines LDAP-Verzeichnisses gibt Auskunft über die eigenen Zertifikate.
- Ein Zeitstempeldienst (TS) kann Daten vertrauenswürdig mit Zeitpunkten verbinden.

Die CA gibt Zertifikate aus und kann diese auch wieder zurückziehen, z. B. mit Hilfe von CRL (Certificate Revocation List) oder über Statusabfragen an den DIR Servcie. Die RA müssen bürgt für die Identität der Zertifikatinhaber, indem sich Identitäten dort ausweisen müssen, um Zertifikate zu erhalten. Meist bieten CAs auch die Erstellung der privaten und öffentlichen Schlüssel als Service an.

6.4.4.2. Hierarchisches Vertrauensmodell

- Bei den beschriebenen Zertifizierungsstellen wird von einem hierarchischen Vertrauensmodell ausgegangen.
- Basis für ein solches System ist eine Wurzelzertifizierungsinstanz (Root-CA) unterhalb derer sich mehrere CAs befinden, welche weiterhin hierarchische untergliedert sein können und von der Root-CA zertifiziert sind.
- Die signierten Benutzerschlüssel können über sie entsprechende Zertifikatskette bis zum Wurzelzertifikat geprüft werden.

Wollen zwei Benutzer das Zertifikat des jeweils anderen prüfen, dann können beide:

- bei der gleichen CA sein und sind somit im Besitz des öffentlichen Schlüssels der Zertifizierungsstelle.
- bei der gleichen Root-CA sein und somit über den Zertifizierungspfad die Prüfung durchführen.
- bei unterschiedlichen Root-CAs, d.h. unterschiedlichen Vertrauensbereichen sein, sodass ein Cross-Zertifikat notwendig ist.

Definition (Cross-Zertifikate):

Ein Cross-Zertifikat baut Zertifizierungspfade über verschiedene CAs unabhängig von der Hierarchie hinweg auf um die Bildung von Zertifizierungspfaden zu ermöglichen. Eine CA_A zertifiziert dabei das Zertifikat einer CA_B.

Statt einer Vertrauenshierarchie, kann auch eine netzartige Vertrauensstruktur etabliert werden: Web of Trust.

6.4.5. Web of Tust

Definition (Web of Trust):

Dem Web of Trust liegt ein Vertrauensnetz zu Grunde, welches auf transitiven Vertrauensbeziehungen beruht.

- Die Funktion die Echtheit eines öffentlichen Schlüssels zu bestätigen, übernehmen hier die Benutzer anstelle einer zentralen Zertifizierungsinstanz.
- Zu jedem öffentlichen Schlüssel anderer Benutzer wird der sog. owner trust festgelegt, welcher beschreibt, wie sehr man den Signaturen des jeweiligen Benutzer vertraut.
- Zusätzlich werden den Schlüsseln eine Menge von Signaturen zugeordnet, welche den öffentlichen Schlüssel zertifizieren (inkl. des Vertrauensgrades (signatory trust)).
- Die Schlüssel können beispielsweise über sog. Key-Server oder als e-Mail Anhänge zugänglich gemacht werden.

Unterschiedliche Vertrauensstufen:

- *ultimate:* eigener Schlüssel
- *complete:* dem Besitzer wird immer vertraut
- *marginal:* dem Besitzer wird normalerweise vertraut
- *untrusted:* dem Besitzer wird nicht vertraut
- *unknown*

Dieses System kommt Beispielsweise bei PGP (Pretty Good Privacy) zum Einsatz.

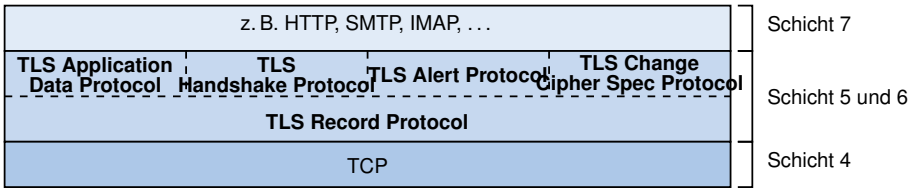
6.5. Sichere Protokolle und Protokollvarianten

6.5.1. TLS

Das Transport Layer Security (TLS) Protokoll ist ein von der IETF standardisiertes Protokoll. In seiner neuesten Version (Version 1.2) ist es durch den RFC 5246 spezifiziert. Ergänzungen dazu finden sich in den RFCs 5746, 5878 und 6176.
TLS setzt auf ein verbindungsorientiertes Transportprotokoll (TCP) auf. Die zu TLS gehörenden Teilprotokolle unterstützen zahlreiche Funktionen. Im Rahmen dieser Vorlesung wird nur ein Teil der Funktionen angesprochen.

- **Handshake Protocol**
  - Aushandlung der Eigenschaften einer Sitzung (Verschlüsselungsalgorithmen, Kompressionsverfahren)
  - Authentifizierung von Instanzen

- **Record Protocol**
  - Fragmentierung der Anwendungsdaten
  - Ver- und Entschlüsselung
  - Kompression und Dekompression



Dienste von TLS
Die in TLS enthaltenen Funktionen dienen zur Realisierung folgender Dienste

- Authentisierung der Instanzen
- Vertraulichkeit der Nachrichten (bzw. der Nutzerdaten)
- Authentisierung und Integritätswahrung der Nachrichten

Diese Dienste werden auf folgende Weise realisiert

- Authentisierung der Instanzen: Bei TLS wird hierzu ein Dialog im Rahmen des TLS Handshake Protocols durchgeführt. Entweder authentisiert sich nur der Server gegenüber dem Client mit Hilfe von Zertifikaten, oder auch (zusätzlich) der Client gegenüber dem Server.
- Vertraulichkeit der Nachrichten: Die ausgetauschten Nutzerdaten werden hierzu mit einem symmetrischen Verschlüsselungsverfahren wie RC4 oder AES verschlüsselt.
- Authentisierung der Nachrichten und Integrität der Nachrichten: Hierzu wird für jede ausgetauschte Nachricht eine kryptografische Hash-Funktion berechnet. In die Berechnung des Hash-Werts fließt ein beiden Seiten bekannter Schlüssel ein.

Das Protokoll TLS kann zum Teil der Sitzungsschicht zugeordnet werden, da es den Kommunikationspartnern erlaubt, einen Sitzungszustand mit gemeinsam bekannten Informationen aufzubauen.
Als Sitzungszustand wird eine Reihe von Informationen (sog. Attribute) verwaltet. Dazu gehören:

- Sitzungsidentifikator (Session ID)
- Zertifikat der Instanz des Kommunikationspartners (Peer Certificate)
- Kryptografische Algorithmen und ihre Parameter (Cypher Suite)
- Funktion zur Kompression (Compression Method)
- Jeweils eine von jeder Instanz generierte Zufallszahl (Seed: ClientHello.random und ServerHello.random)

- Mehrere Schlüssel zur Verschlüsselung und zur Nachrichtenauthentisierung. Zunächst wird ein Premaster-Secret ausgehandelt. Daraus werden dann weitere Schlüssel abgeleitet.

Hinweis: TLS verwendet abhängig von den zur Verfügung stehenden Authentifizierungsmethoden unterschiedliche Handshakes. Im folgenden betrachten wir den TLS Simple Handshake mit serverseitigem Zertifikat.

TLS Handshake Protocol

1. **Client Hello**
   Unterstützte TLS-Versionen, unterstützte Verschlüsselungsprotokolle und Kompressionsmethoden, Zufallszahl (Seed)

2. **Server Hello**
   Gewählte TLS-Version, gewählte Verschlüsselungs- und Kompressionsmethoden, Zufallszahl

3. **Server Certificate**
   Enthält den öffentlichen Schlüssel des Servers

4. **Server Hello Done**
   Ende des serverseitigen Handshakes

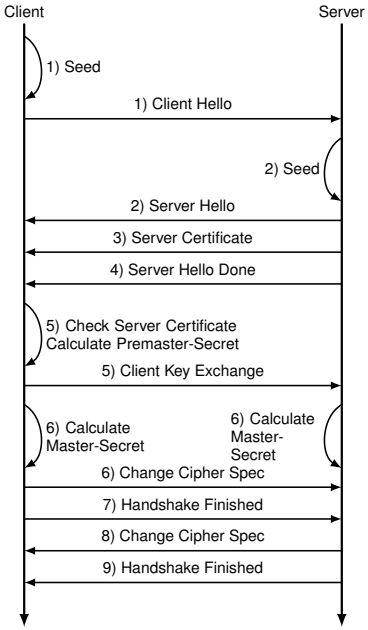
5. **Client Key Exchange**
   Mit RSA asymmetrisch verschlüsseltes Premaster-Secret, aus dem zusammen mit den Seeds ein gemeinsames Master-Secret für die symmetrische Verschlüsselung abgeleitet wird

6. **Change Cipher Spec**
   Letzte unverschlüsselte Nachricht des Clients

7. **Handshake Finished**
   Erste verschlüsselte Nachricht des Clients

8. **Change Cipher Spec**
   Letzte unverschlüsselte Nachricht des Servers

9. **Handshake Finished**
   Erste verschlüsselte Nachricht des Servers



Anmerkungen zum TLS-Handshake

- Der hier gezeigte TLS-Handshake ist der sog. Simple TLS Handshake. Verfügt auch der Client über ein Zertifikat und wird dies vom Server angefordert, so gibt es einen modifzierten Client-Authenticated TLS Handshake.
- Bei Wiederaufnahme einer Session enthalten Client- und Server-Hello eine Session-ID, mit der die zuvor unterbrochene Session identifiziert werden kann. Hierfür gibt es einen abgekürzten Resumed TLS Handshake.
- TLS arbeitet mit unterschiedlichen Anwendungsprotokollen zusammen. Diese werden dann häufig durch ein angehängtes „S“ gekennzeichnet, z. B. HTTPS.

Die Themen

- symmetrische Verschlüsselung,
- asymmetrische Verschlüsselung und
- Public Key Infrastructures

werden detaillierter in der Vorlesung Netzsicherheit (Wintersemester) behandelt.

TLS Record

- Das TLS Record Protocol kapselt die Nutzdaten höherer Ebenen als TLS Record.

- Die Nutzdaten sind durch die kryptografische Hashsumme integritätsgeschützt.
- Data + Hash + pad + pad length sind verschlüsselt.

6.5.2. HTTPS

6.5.2.1. HTTPS - Funktionsweise

Kommunikationen über HTTP können transparent durch TLS gesichert werden.

- Zur Unterscheidung von HTTP findet HTTPS über Port 443 statt.
- Browser nutzen hierfür das Schema https://

  Dieses Schema wurde seit 1995 von Netscape in seinen Browsern zur Kennzeichnung von HTTP über SSL verwendet. Dies führte zu der heute gebräuchlichen Bezeichnung.

Typischerweise werden die Authentizität des kontaktierten Servers sichergestellt sowie die kommunizierten Daten vor dem Austausch verschlüsselt und vor Manipulationen geschützt.

- Aufgrund der Kapselung werden insbesondere auch die HTTP-Header verschlüsselt übertragen. Es ist somit lediglich ersichtlich, mit welcher IP-Adresse über welchen Port Daten ausgetauscht werden.
- Zur Authentifikation des Servers muss die Zertifikatskette überprüft werden. In den meisten Browsern ist hierfür standardmäßig eine Vielzahl an gängigen Root-Zertifikaten hinterlegt.

  Für alle weiteren gültigen, aber nicht verifizierbaren Zertifikate bekommt man in der Regel eine Sicherheitswarnung vom Browser. Ohne die Authentifikation des Kommunikationspartners ist auch die Verschlüsselung teilweise hinfällig, da nicht ausgeschlossen werden kann, direkt mit einem Angreifer zu kommunizieren

6.5.2.2. CVE-2014-0160 (Heartbleed)

Heartbeat: Keep-Alive-Extension für TLS (RFC 6520; Februar 2012)

- **Request**: Client sendet beliebigen Text + Längenangabe desselben
- **Response**: Server soll mit demselben Text antworten

Exploit:

- Client sendet Text + zu große Längenangabe
- Server antwortet mit Text plus den im Speicher dahinter liegenden Daten (Read overrun)
- Das Längenfeld besitzt 2 Byte ⇒ pro Request sind maximal 2^{16} Byte = 64KB auslesbar.

Fehler:

- OpenSSL-Implementation vertraute auf Längenangabe des Clients
- Bug wurde beim Code-Review nicht entdeckt
- Längenfeld u. U. länger als notwendig
- OpenBSD Speicherschutzmechanismen waren bei OpenSSL deaktiviert

Originalcommit bei GitHub:
https://github.com/openssl/openssl/commit/4817504d069b4c5082161b02a22116ad75f822b1

6.5.3. Sichere E-Mail

6.5.3.1. Sichere E-Mail - Funktionsweise

Ohne weiteres Zutun werden E-Mails im Klartext versandt.
Somit können sie während des Transports und der (Zwischen-)Speicherung eingesehen und manipuliert werden. Daher sind zur Wahrung der Vertraulichkeit, der Authentizität und der Integrität auch hier Zusatzmaßnahmen notwendig.

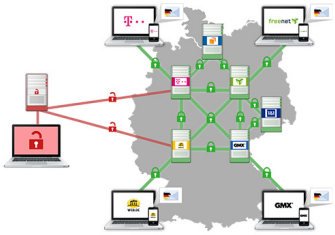
6.5.3.2. Einschub: E-Mail made in Germany

- Initiative von 1&1 und Telekom
- Freenet und Strato sind weitere Teilnehmer
- Umgesetzt in 2013

Auszüge aus der Beschreibung der Initiative:
. . . Ihre Daten werden automatisch verschlüsselt. Um das Mitlesen Ihres E-Mail-Verkehrs im Internet zu verhindern. . .
. . . Ihre Daten werden immer SSL-verschlüsselt übermittelt und so vor dem Zugriff von Dritten geschützt. . .

. . . Um eine sichere Übertragung auf allen Übertragungswegen zu gewährleisten, werden auch zwischen den E-Mail-Servern von freenet, GMX, Telekom und WEB.DE alle Daten ausschließlich verschlüsselt übertragen. . .

. . . De-Mail geht noch weiter: Auf Grundlage der De-Mail Gesetze entwickelt, gewährleistet De-Mail . . . zusätzlich die einwandfreie Identität von Sender und Empfänger. . .



Quelle: http://www.e-mail-made-in-germany.de/Verschluesselung.html

De-Mail-Gesetz (2011 in Kraft getreten): De-Mail-Dienste, "die einen sicheren, vertraulichen und nachweisbaren Geschäftsverkehr für jedermann im Internet sicherstellen sollen" (§1, De-Mail-G)
Vorgehensweise:

- Kunde liefert einmalig Identitätsnachweis gegenüber Mailanbieter
- Login via Two-Factor-Auth ("sicher") oder Username/Passwort ("unsicher")
- Verfassen von Nachrichten via Webinterface
- Verschlüsselung und Signatur NACH Erhalt durch Mailanbieter
- Versand via SMTPS (RFC 3207 von 2002)

Probleme:

- Keine Ende-zu-Ende-Verschlüsselung
- Elektronische Signatur, jedoch nicht beim Absender, sondern durch Mailanbieter
- Serverseitige Entschlüsselung der e-Mails (Argument: Virenscan)

⇒ weiterhin Vertrauen in Mailanbieter notwendig ⇒ Aussagekraft der Signatur massiv gemindert

6.5.3.3. Sichere E-Mail - Fortsetzung

Ohne weiteres Zutun werden E-Mails im Klartext versandt.
Somit können sie während des Transports und der (Zwischen-)Speicherung eingesehen und manipuliert werden. Daher sind zur Wahrung der Vertraulichkeit, der Authentizität und der Integrität auch hier Zusatzmaßnahmen notwendig.
Wird TLS zwischen Client und Mailserver sowie zwischen den Mailservern eingesetzt, so wird die Nachricht dadurch vor Dritten geschützt. Eine Einsicht und Manipulation durch zuständige Mail-Relay-Server ist jedoch weiter möglich. Darüber hinaus wird die Authentizität des Absenders nie verifiziert.
Zur tatsächlichen Wahrung dieser Schutzziele müssen die Verschlüsselung und Mechanismen zur Authentifikation **bereits beim Client** durchgeführt werden.
Zwei verbreitete Technologien hierfür sind

- S/MIME
- OpenPGP

6.5.3.4. S/MIME & OpenPGP

Das S/MIME (RFC 5750 und 5751) und OpenPGP (RFC 4880) sind Standards, den den Versand signierter und verschlüsselter Nachrichten ermöglichen.

Beide sind ein hybrides Public-Key-Verfahren. Beim Versand einer E-Mail wird diese zuerst signiert und dann mit einem zufälligen Einmalschlüssel (symmetrisch) verschlüsselt. Dieser wird dann asymmetrisch mit dem Public Key des Empfängers verschlüsselt. Beide Informationen werden dann zusammen versandt.

Unterschied zwischen beiden liegt im Trust Model:

- S/MIME nutzt hierarchische X.509 Zertifizierungsinfrastruktur
- OpenPGP verwendet ein Web of Trust-Modell

S/MIME nutzt die hierarchische X.509-Zertifizierungsinfrastruktur. Die verwendeten Schlüssel werden daher von einer CA beglaubigt. Zur Authentifikation des Senders von signierten Nachrichten ist auch hier die Überprüfung der gesamten Zertifikatskette notwendig. Aktuelle Mailprogramme haben hierzu gängige Root-Zertifikate hinterlegt. Im Gegensatz zu S/MIME verwendet OpenPGP keine zentrale Schlüsselinfrastruktur. Anstelle dessen ist jeder in der Lage, die Schlüssel anderer zu beglaubigen. Je engmaschiger dieses Netz gegenseitiger Beglaubigungen (Web of Trust) wird, als desto vertrauenswürdiger können die beteiligten Schlüssel angesehen werden.
Die Einbettung der verschlüsselten beziehungsweise signierten Daten in e-Mails wird MIME verwendet. Es existieren dafür entsprechende Content-Types:

- multipart/signed
- multipart/encrypted
- multipart/pkcs7-mime

Mögliche Verschlüsselungs- und Signaturalgorithmen sind im RFC spezifiziert. RSA beziehungsweise RSA mit SHA-256 werden bei S/MIME als obligatorischer Standard vorgegeben. Bei OpenPGP werden beispielsweise CAST5, BLOWFISH, AES für Erstere und MD5, SHA1 sowie SHA512 für Letzeres unterstützt.
Auch wenn die Ziele und teilweise sogar die verwendeten Algorithmen identisch sind, unterscheidet sich dennoch das Datenformat. Daher sind S/MIME und OpenPGP zueinander nicht kompatibel.

6.5.3.5. Format einer mit S/MIME signierten Nachricht

```
...
Subject: ...
Content-Type: multipart/signed; protocol="application/pkcs7-signature";
micalg=sha1; boundary="------------ms020508090406050008090703"

This is a cryptographically signed message in MIME format.

--------------ms020508090406050008090703
```

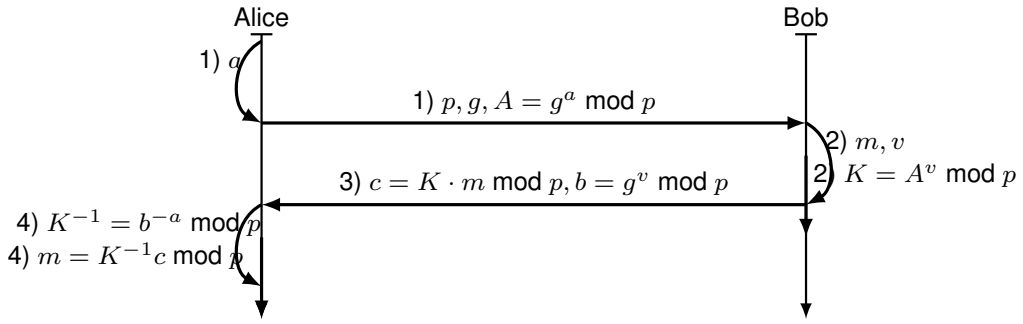
```
Content-Type: text/plain; charset=UTF-8
Content-Transfer-Encoding: quoted-printable

Hello, ...

--------------ms020508090406050008090703
Content-Type: application/pkcs7-signature; name="smime.p7s"
Content-Transfer-Encoding: base64
Content-Disposition: attachment; filename="smime.p7s"
Content-Description: S/MIME Cryptographic Signature

MIAGCSqGSIb3DQEHAqCAMIACAQExCzAJBgUrDgMCGgUAM...
--------------ms020508090406050008090703--
```

6.5.3.6. Format einer mit GPG verschlüsselten Nachricht

```
...
Subject: ...
X-Enigmail-Version: 1.6
Content-Type: multipart/encrypted;
protocol="application/pgp-encrypted";
boundary="sReSwidkN7wigApS7WvmWHOwTRXFRXCpI"

This is an OpenPGP/MIME encrypted message (RFC 4880 and 3156)
--sReSwidkN7wigApS7WvmWHOwTRXFRXCpI
Content-Type: application/pgp-encrypted
Content-Description: PGP/MIME version identification

Version: 1

--sReSwidkN7wigApS7WvmWHOwTRXFRXCpI
Content-Type: application/octet-stream; name="encrypted.asc"
Content-Description: OpenPGP encrypted message
Content-Disposition: inline; filename="encrypted.asc"

-----BEGIN PGP MESSAGE-----
Version: GnuPG v1.4.14 (GNU/Linux)
Comment: Using GnuPG with Thunderbird - http://www.enigmail.net/

hQEMAOtQHerC/rXTAQf...
-----END PGP MESSAGE-----

--sReSwidkN7wigApS7WvmWHOwTRXFRXCpI--
```

ElGamal-Verschlüsselung

Das ElGamal-Verschlüsselungsverfahren basiert auf dem bereits vorgestellten Diffie-Hellman-Verfahren:



1. Alice berechnet mithilfe der primitiven Kongruenzwurzel g zur Primzahl p ihren öffentlichen Schlüssel A und veröffentlicht ihn zusammen mit der Werten für g und p.
2. Bob möchte nun Nachricht m an Alice schicken. Hierzu erzeugt er eine Zufallszahl v.
3. Bob berechnet nun den Chiffretext $c = A^v \cdot m$ mod p und verschickt ihn zusammen mit $b = g^v$mod p
4. Alice kann nun die Inverse des Schlüssels berechnen und damit aus c wieder die Nachricht m gewinnen.

6.5.4. DNSSEC

DNSSEC ist eine Erweiterung des DNS-Protokolls um Authentizitäts- und Integritätsfunktionen. Die aktuelle Version dieser Erweiterung wurde 2005 in RFC 4033, 4034 und 4035 spezifiziert.

Schutzziele in DNSSEC für DNS-Daten

- Authentizität
- Integrität
- Nicht: Vertraulichkeit

Ein Angreifer kann DNS-Daten nicht mehr manipulieren (also z. B. die Manipulation der Namensauflösung für die Vorbereitung eines MitM-Angriffs) aber immer noch sehen, welche Namen angefragt werden.

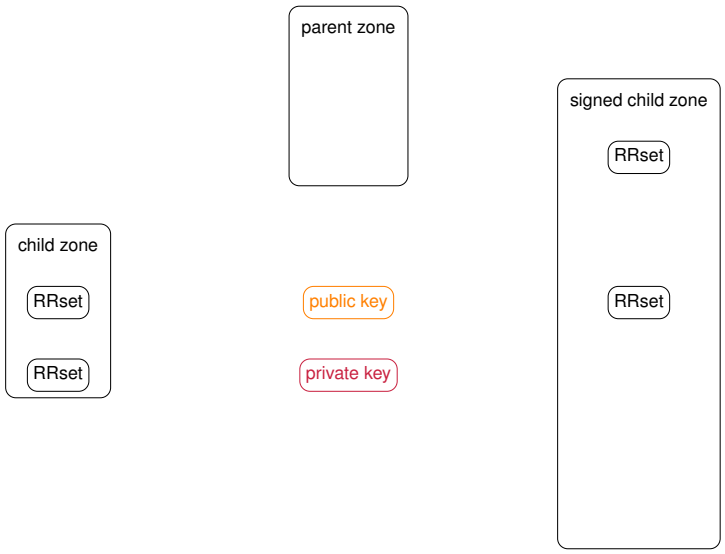
Integration

- DNSSEC erweitert das DNS-Protokoll direkt, es gibt keinen zusätzlichen Security Layer.
- Abwärtskompatibel: bestehende Systeme funktionieren weiterhin, wenn auch ohne die zusätzliche Sicherheit von DNSSEC.
- DNSSEC sichert zunächst nur die Namensauflösung, nicht jedoch die darauf aufbauenden Verbindungen.
- **Zusätzliche DNS Resource Records für DNSSEC**
  - DNSKEY – öffentlicher Schlüssel
    Enthält den öffentlichen Schlüssel für die Signaturüberprüfung. Der private Schlüssel wird natürlich nicht veröffentlicht.

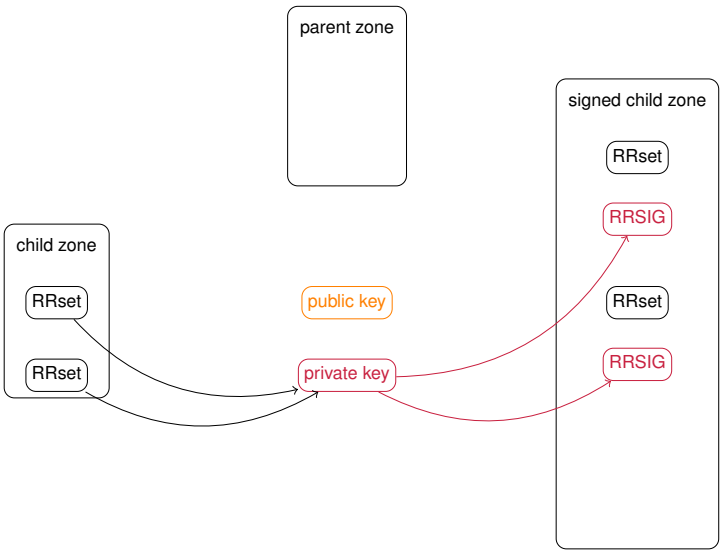
  - RRSIG – Signatur
    Enthält die Signatur über die Daten eines Resource Record Set (RRSet), z.B. IPv4-Addressen.
  - DS – Hash über ein DNSKEY
    Der Hash (zusammen mit der zugehörigen RRSIG) stellt die Signaturkette zu einem DNSKEY in einer untergeordneten Zone sicher.
  - NSEC – Nachweis der Nicht-Existenz von Namen
    Ein NSEC Record (mit entspechender RRSIG) wird verwendet um nachzuweisen, dass ein Domain Name nicht existiert. Der NSEC Record enthält den lexikographisch vorhergehenden und nachfolgenden Namen und kann somit beweisen, dass zwischen diesen beiden Namen keine weiteren Namen existieren. NSEC Records geben die Löcher zwischen den Domain Names an.

- Die DNS Zone signiert die Daten der DNS Resource Records (z.B. die IPv4-Addressen für den Record) mit dem privaten Schlüssel.
- Diese Signatur (RRSIG) wird zusammen mit den jeweiligen DNS Resource Records zurückgeliefert.
- Ein Resolver verifiziert die Signatur mit dem öffentlichen Schlüssel der Zone (DNSKEY).
- Der öffentliche Schlüssel wird von der darüberliegenden Zone durch den DS Record validiert.
- Der Hash (DS) des öffentliche Schlüssel der Root Zone ist bekannt.
- Damit eine Zone DNSSEC signiert werden kann, müssen alle darüberliegenden Zonen bis zur Root Zone signiert sein.
- Jede Zone kann nur die eigenen Records signieren.

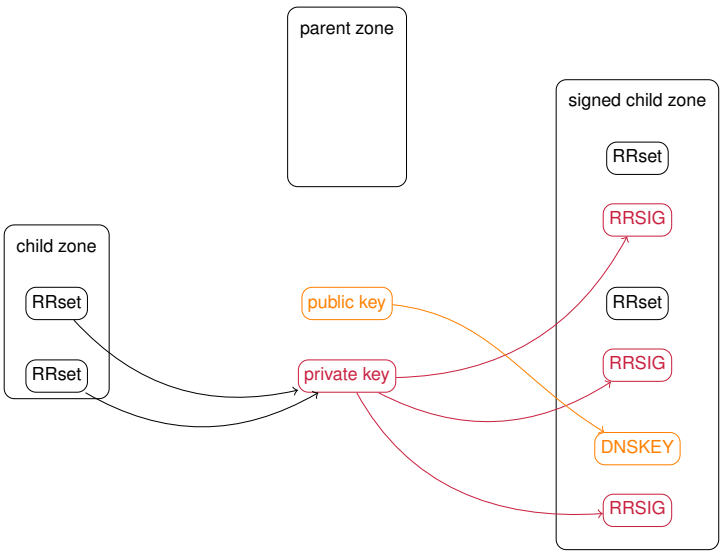
6.5.4.1. DNSSEC Zone Signing



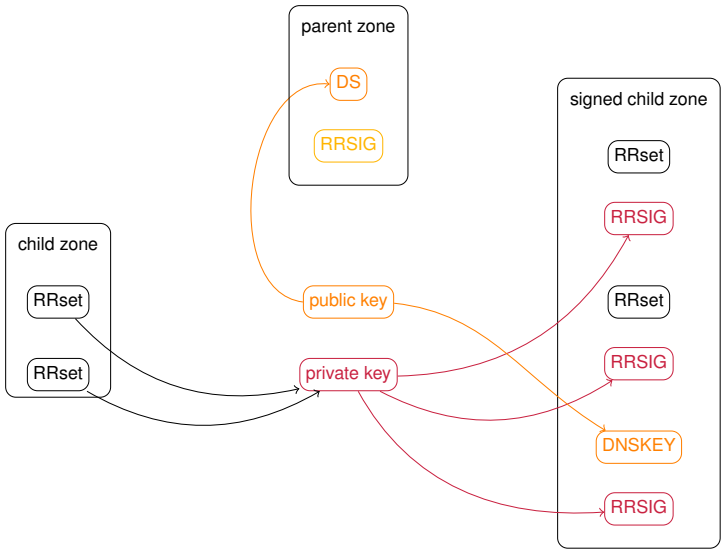
Die Zone “child zone” möchte die eigenen Daten (RRSets, wie z. B. IP-Addressen oder Mail Server Einträge) signieren und über DNSSEC sichern. Hierzu wird zunächst ein Schlüsselpaar erstellt.



Die Daten (RRsets) werden mit dem privaten Schlüssel signiert. Die Signaturen werden in den RRSIGs gespeichert. Diese RRSIGs werden mit dem jeweiligen RRset zurückgeliefert.



Der öffentliche Schlüssel wird im DNSKEY Record in der Zone gespeichert. Er wird für die Signaturüberprüfung benötigt. Zusätzlich wird auch eine Signatur über den DNSKEY angelegt und gespeichert.



Der Hash des öffentlichen Schlüssels (DS Record) wird auf sicherem Weg in der darüberliegenden Zone eingetragen. Dieser DS Record wird von der darüberliegenden Zone signiert. Dadurch wird eine Chain-of-Trust bis in die Root Zone aufgebaut.

6.5.4.2. Implementierung DNSSEC

- Die Root Zone ist seit Juli 2010 signiert.
- Die `de` Zone ist seit Mai 2011 signiert.
- Der DNSKEY wird oft in ein Key Signing Key (KSK) und ein Zone Signing Key (ZSK) aufgesplittet.
  - Der KSK ist länger (2048 bit RSA) und länger gültig (2–4 Jahre).
  - Der DS dieses Schlüssels wird in der übergeordneten Zone eingetragen.
  - Der ZSK ist kürzer (512 bit RSA) und nur 1–2 Monate gültig.
  - Der ZSK wird mit dem KSK signiert.
  - Die RRSIGs werden mit dem ZSK erstellt.
  - Dies wird gemacht, damit die Signaturen nicht zu lang werden und auch schneller zu verifizieren sind.
- NSEC3 verwendet für den Nachweis der Nicht-Existenz von Namen deren Hashs und kann somit verhindern, dass alle Namen in der Zone sichtbar werden.
- Der Client vertraut seinem DNS Resolver. Der (rekursive) Resolver führt die eigentliche DNSSEC-Überprüfung durch. Problem: Wie bekommt der Client die Antwort vom Resolver?
- DNSSEC sichert nicht die Übertragung zwischen zwei Endsystemen.
- DNSSEC kann aber verwendet werden um Schlüsselmaterial für eine solche Verbindung sicher zu verteilen:
  - TLSA Resource Records (RFC 6698) speichern den Hash eines X.509 Zertifikats, sodass DNSSEC als Trust Anchor für TLS verwendet werden kann.

– SSHFP Resource Records (RFC 4255) speichern den SSH Server Fingerprint im DNS.

6.6. Netzwerkmonitoring und -schutz

6.6.1. Motivation

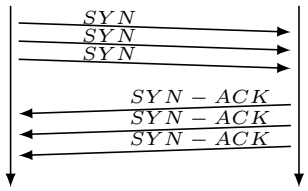
- Reichen bisherige Maßnahmen aus?
- Nein, leider nicht...
- Fehlerhafte Implementierungen, Fehlkonfigurationen oder falsche Benutzung können zu Schwachstellen führen, welche für Angriffe ausgenutzt werden können.
- Zudem zielen einige Angriffe auf bisher nicht betrachtete Schutzziele ab (z.B. Availability).

Beispiel: Denial-of-Service durch SYN-Flooding

Definition (Denial-of-Service):

Unter dem Begriff Denial-of-Service werden Angriffe zusammengefasst, welche durch gezielt erzeugte Überlastsituationen Netzkomponenten in deren Erreichbarkeit komplett oder wenigstens teilweise einschränken.

- Der Angreifer sendet TCP-SYN Paketen an ein Opfer.
- Das Opfer antwortet mit einem SYN-ACK und hält die Verbindung einseitig offen, da kein weiteres ACK eintrifft.
- Um den Zustand zu halten müssen Systemressourcen eingesetzt werden.
- Reguläre Anfragen können im schlimmsten Fall nicht mehr angenommen werden, und das Opfer ist nicht mehr erreichbar.

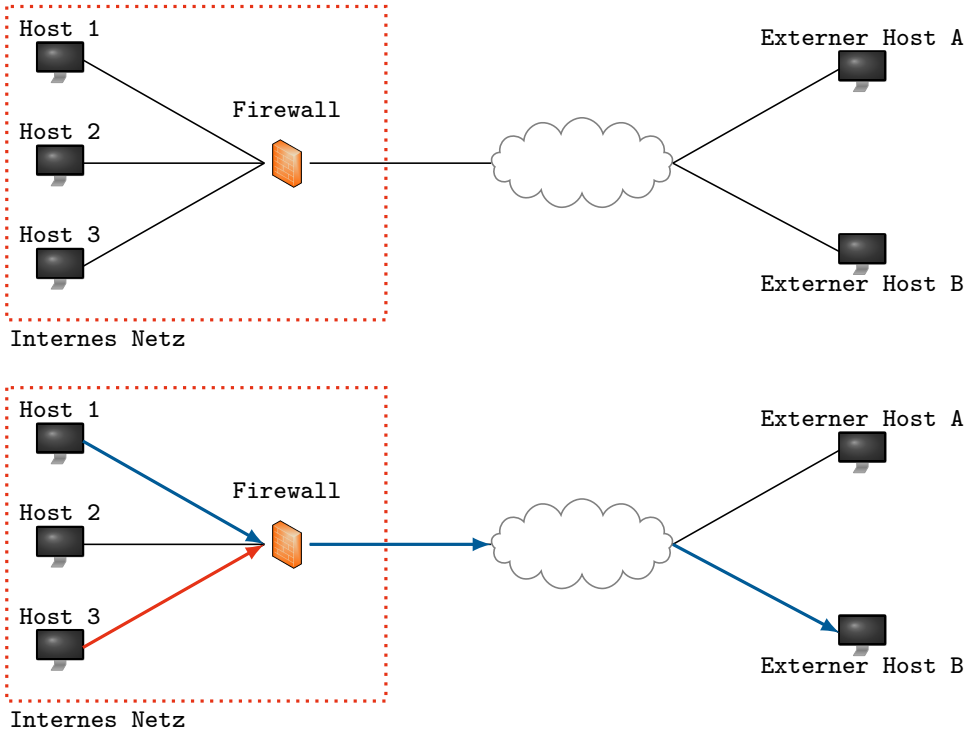


Ein Schutz des Netzwerks im laufenden Betrieb durch spezielle Komponenten (z.B. Firewalls) als auch das stete Überwachen des Netzes (Netzwerkmonitoring) sind daher unerlässlich.

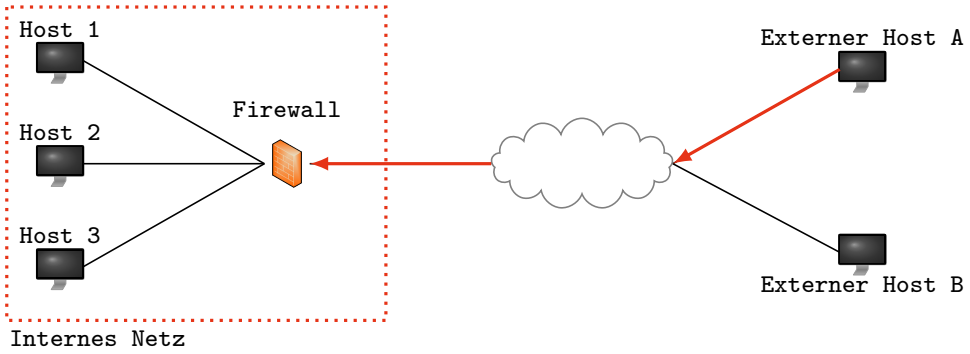
6.6.2. Firewall

- Firewalls schotten unterschiedliche Netze voneinander ab und kontrollieren die Übergänge in diese Netze, z.B. vom Internet in private Netze.
- Sie dienen als zentraler Kontrollpunkt, an welchem entschieden wird, welcher Verkehr von außen nach innen und umgekehrt weitergeleitet wird.

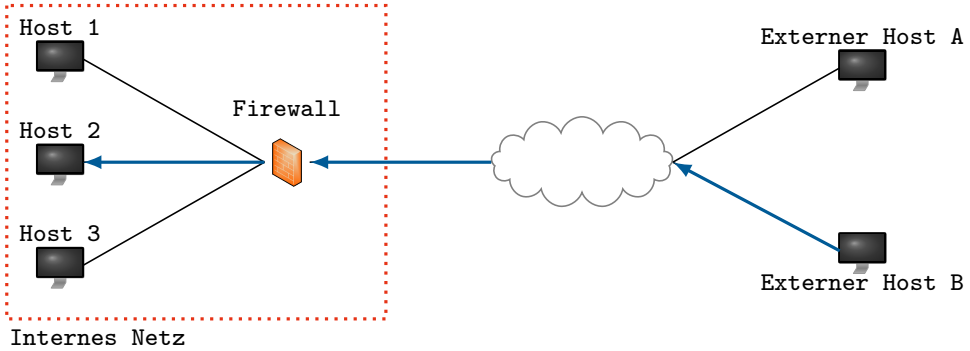
- Es kann sich dabei sowohl um Soft- als auch Hardwarekomponenten handeln.



- Pakete aus dem internen Netz können geblockt oder weitergeleitet werden.



- Analoges geschieht bei Paketen aus dem externen in das interne Netz. Es können Pakete geblockt werden.



- Oder Pakete können weitergeleitet werden.

Beispiel - ufw (Uncomplicated Firewall) für Ubuntu

```
### RULES ###
### tuple ### allow tcp 21 0.0.0.0/0 any 0.0.0.0/0 in
-A ufw-user-input -p tcp --dport 21 -j ACCEPT
### tuple ### allow any 123 0.0.0.0/0 any 0.0.0.0/0 in
-A ufw-user-input -p tcp --dport 123 -j ACCEPT
-A ufw-user-input -p udp --dport 123 -j ACCEPT
### tuple ### deny tcp 25 0.0.0.0/0 any 0.0.0.0/0 in
-A ufw-user-input -p tcp --dport 25 -j DROP
### tuple ### allow tcp any 0.0.0.0/0 25 131.87.63.9 in
-A ufw-user-input -p tcp -s 131.87.63.9 --sport 25 -j ACCEPT
### END RULES ###
```

Wie kann man das selbst mal ausprobieren?

- Bei Ubuntu in den Paketquellen (repositories), *iptables* ist nötig
- *ufw* bietet lediglich eine vereinfachte Bedienung zur Erstellung der *iptables*-Regeln
- Angelegte Regeln sind dann unter */var/lib/user.rules* oder */lib/ufw/user.rules* (siehe Ausschnitt oben)
- Eine Anleitung zum aktivieren der Firewall und zur Erstellung von Regeln ist unter: *wiki.ubuntuusers.de/ufw* zu finden

Nach welchen Kriterien kann gefiltert werden?

- Paketfilter können auf verschiedenen Schichten ansetzen
- Transportprotokoll (TCP, UDP)
- Portnummern
- IP-Adressen (sowohl Ziel- als auch Senderadressen)
- Services und Applikationen (ftp, http, smtp, ...)

Wie kann eine Firewall reagieren?

- ALLOW - Paket weiterleiten
- DENY - Paket verwerfen
- REDIRECT - Paket umleiten

Arten von Firewalls

- Zustandslos: jedes Paket wird einzeln betrachtet
- Zustandsbehaftet: Pakete können in Beziehung gesetzt werden (z.B. für TCP SYN-/ACK)

Ist das Netz nun sicher?

- Eine Firewall schützt das Netz nur vor Angriffen von außen.
- Bedrohungen können allerdings auch von innen kommen (empfehlenswert: Arte Netwars Dokumentation).

6.6.2.1. Intrusion Detection Systeme (IDS)

- Ist es nicht möglich, Angriffen vorzubeugen oder sie zu verhindern, dann ist es nötig, sie zu erkennen und im Anschluss entsprechende Maßnhamen zu ergreifen (z.B. Mitigation, Recovery, ...).

Definition (Intrusion und Intrusion Detection):

Als Intrusion bezeichnet man eine Abfolge zusammengehöriger Aktionen eines Angreifers mit der Absicht, ein Zielsystem zu kompromittieren.
Intrusion Detection bezeichnet den Prozess zur Identifizierung solcher schadhaften Aktionen.

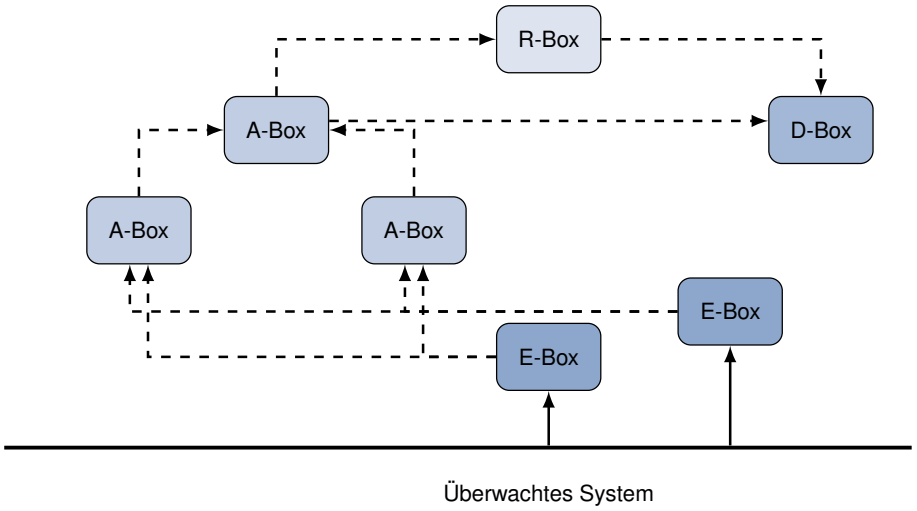
Anforderungen an ein IDS:

- Genauigkeit, d.h. eine geringe False-Positive-Rate
- Performanz, um möglichst in Echtzeit arbeiten zu können
- Vollständigkeit, d.h. geringe False-Negative-Rate
- Fehlertolerant, d.h. resistent gegenüber Angriffen auf das IDS selbst
- Skalierbar, d.h. auch eine große Anzahl eingehender Events muss bearbeitet werden können ohne Informationen zu verlieren

6.6.2.2. CIDF Architektur

- Ein IDS kann mit Hilfe des Common Intrusion Detection Framework (CIDF) [5] umgesetzt werden.
- Es werden vier Komponenten und deren Rollen festgelegt.

[5] http://gost.isi.edu/cidf/drafts/architecture.txt



Erläuterungen zum CIDF

- **Event-Boxen (E-Box)** generieren Events, welche aus dem Monitoring gewonnen werden können.
- **Analyse-Boxen (A-Box)** analysieren die Events der E-Boxen. Das Resultat einer A-Box ist typischerweise ein Alarm, welcher von weiteren A-Boxen weiterverarbeitet werden kann. Diese arbeiten dann auf einem höheren Abstraktionslevel, beispielsweise Korrelation oder Aggregation von Alarmen.
- **Datenbank-Boxen (D-Box)** speichern Events und Alarme und dienen als Grundlage für Forensik und die persistente Haltung der Daten.
- **Response-Boxen (R-Box)** führen Aktionen gegen erkannte Angriffe aus. Eine mögliche Aktion ist die Umkonfiguration einer Firewall.

Hinweis: Der term Box steht hierbei für eine einzelne Komponente.

6.6.2.3. IDS Taxonomie

Erkennungsmethode

- *Anomalie-basierte Erkennung:* Das Normalverhalten des Systems wird dabei zu Grunde gelegt und mit dem aktuellen Verhalten abgeglichen. Bei Abweichungen spricht man von einer Anomalie. Dies kann allerdings sowohl ein Angriff, eine Fehlkonfiguration oder lediglich unerwartetes Verhalten sein. Letzteres zeigt die Probleme der Anomalie-Erkennung auf, da eine Vorabentscheidung über den Normalzustand meist nur schwer oder gar nicht möglich ist.
- *Signatur-basierte Erkennung:* Hierbei werden bekannte Angriffe spezifiziert und das System mit Hinblick auf diese vorgegebenen Muster untersucht. Nicht bekannte Angriffe können auf diese Weise allerdings nicht erkannt werden.

Lokation der Überwachung

- *Host-basierte Erkennung:* Das IDS befindet sich hierbei auf dem zu überwachenden Host. Dadurch wird zwar ein tieferer Einblick in das Hostsystem selbst möglich, allerdings ist keine globale Netzwerksicht möglich.
- *Netzwerk-basierte Erkennung:* Das IDS befindet sich hier im Netzwerk und kann den gesamten Verkehr überwachen. Detaileinblicke auf Vorgänge einzelner Hosts sind hier nicht möglich.

Um beide Vorteile zu verbinden werden sog. hybride IDS eingesetzt, welche dann allerdings untereinander koordiniert und die anfallenden Informationen miteinander kombinieren müssen.

6.6.2.4. IDS Beispiele

SNORT - signaturbasiertes Network IDS

- `alert tcp any any -> any 80 (content:! "GET";)` → Alle TCP-Verbindungen auf Port 80 die kein „GET “ enthalten, provozieren einen Alarm
- `alert tcp any any -> any 21 (msg:"FTP ROOT" content:"USER root"; nocase;)` → Ein Root-Login Versuch über FTP wird als Angriff erkannt
- `alert tcp any any -> any 80 (content:"foo"; content:"evil"; http_cookie;)` → Sollte in einem Cookie der String „evil “ enthalten sein, wird ein Alarm geworfen (statt cookie kann auch header verwendet werden)

BRO

- Skripte für Anomalie-Erkennung können definiert werden
- Beispiel: Erkennung eines FTP Brute-Force Angriffs (BRO Manual)
- Dies basiert auf der Überschreitung der zulässigen Schwellwerte für Fehlversuche beim Login innerhalb einer bestimmten Zeitspanne
- Es werden sowohl falsche Passwörter als auch Nnutzernamen betrachtet
- Für diese Analysen kann auch auf Logdaten der Rechner zurückgegriffen werden

7. Verteilte Systeme

7.1. Motivation

7.1.0.5. Was ist ein verteiltes System?

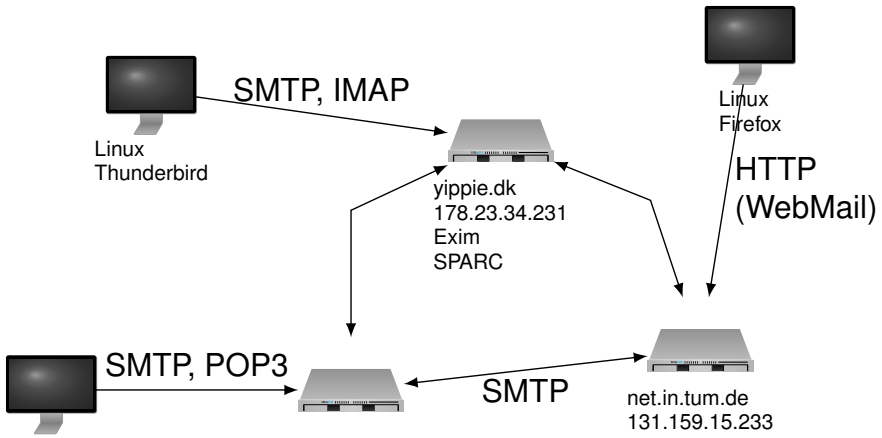
Definition:

„Ein verteiltes System ist eine Menge voneinander unabhängiger Computer, die dem Benutzer wie ein einzelnes, kohärentes System erscheinen.“ [13]

Warum verteilte Systeme?

- Skalierbarkeit: Viele Rechner stellen im Verbund mehr Ressourcen zur Verfügung.
  - *Vorteil*: Der Durchsatz des Gesamt-Systems steigt.
  - *Nachteil*: Der Overhead, insbesondere durch Kommunikation, steigt, sodass die Effizienz abnimmt. Bei zu kleinen Problemen oder zu hohem Overhead kann der Gesamtdurchsatz abnehmen.
- Ausfallsicherheit: Der Dienst als Ganzes kann auch noch angeboten werden, wenn ein einzelner Rechner ausfällt.
  - *Vorteil*: Die Zuverlässigkeit des verteilten Dienstes steigt gegenüber einem zentralisierten Dienst.
  - *Nachteil*: Die Ausfallwahrscheinlichkeit steigt mit der Anzahl der Komponenten eines Systems. Bei einem zentralisisierten Dienst ist oft weniger Ausfallsicherheit notwendig (Backups, Hot-Standby).
- Heterogenität: Rechner mit verschiedener Hardware oder unterschiedlichen Eigentümern können einen gemeinsamen Dienst anbieten.

7.1.0.6. Beispiel: E-Mail



7.1.0.7. Beispiel: E-Mail

- E-Mail ist ein verteilter Dienst:
  - Mehrere Hosts nehmen teil, insbesondere mehrere Server.
  - Server und Clients können unterschiedlichen Parteien gehören.
- E-Mail ist heterogen:
  - Netzwerkanbindung
    z. B. Schicht 2: Ethernet, WLAN, ...
    z. B. Schicht 3: IP, IPX, ATM, ...
  - Hardware
    z. B. Architektur: x86, x64, ARM, SPARC, ...
    z. B. Endianness: Little Endian, Big Endien
  - Software
    z. B. Betriebssystem: GNU/Linux, Windows, MacOS, ...
    z. B. Mail Transfer Agent: Postfix, Exim, MS Exchange Server, ...
    z. B. User Agent: Thunderbird, mutt, MS Outlook, ...
  - Protokolle
    z. B.: SMTP, SMTP+TLS, POP3, IMAP, WebMail, ...
- Kommunikation zwischen allen Teilnehmern ist dank standardisierter Protokolle trotzdem möglich.

7.1.0.8. Fokus

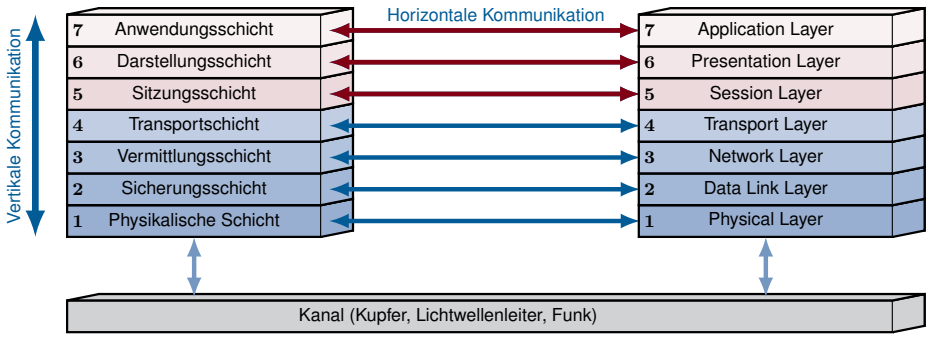
Es gibt zahlreiche Problemfelder im Bereich der Verteilten Systeme, zum Beispiel:

- Entwurf von Kommunikationsprotokollen (vgl. Kapitel 5)
- Abstraktion vom Basissystem durch sog. "Middleware"
  - Kommunikation
  - Persistenter Datenspeicher
- Migration/Mobilität von Systemkomponenten
- Fehlertoleranz
- Skalierbarkeit

Lernziel dieser Vorlesung:

- Probleme Verteilter Systeme verstehen und klassifizieren zu können
- Methoden zur Implementierung Verteilter Systeme am Beispiel bestehender Lösungen verstehen

7.1.0.9. Einordnung im ISO/OSI-Modell



7.2. Homogene, skalierbare Paradigmen

7.2.1. MPI

7.2.1.1. Problemstellung

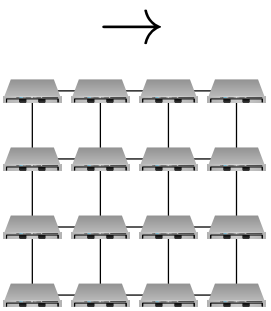
Szenario: Ein Super-Computer besteht aus 10 000 Knoten. Alle Knoten verfügen über die gleiche Hardware und sind in einem regelmäßigen Muster in einem gemeinsamen Netzwerk angeordnet. Dies ist typisch im Bereich des High-Performance Computing (HPC).

Problem: Wie kann man reguläre Datenstrukturen, wie z. B. Matrizen, effizient parallel verarbeiten?

Idee: Man kann die Datenstruktur auf die Topologie des Rechnernetzes abbilden.

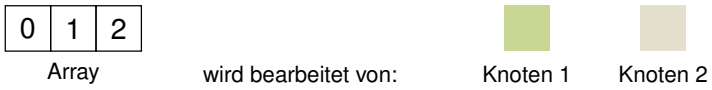
$$\begin{bmatrix} 1.0 & 0.7 & 0.4 & \cdots & 0.0 \\ 0.6 & 1.0 & 0.7 & \cdots & 0.1 \\ 0.5 & 0.8 & 1.0 & \cdots & 0.1 \\ 0.3 & 0.5 & 7.0 & \cdots & 0.1 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0.0 & 0.0 & 0.1 & \cdots & 1.0 \end{bmatrix}$$

Matrix

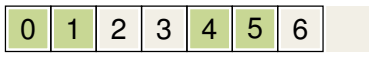


HPC-Cluster

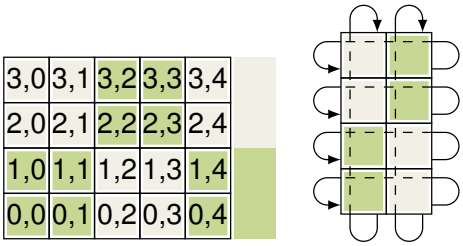
7.2.1.2. Beispiel: Abbildung einer Matrix auf Knoten



- **1-dimensional**
  z. B. Vektor mit Elementen vom Typ (Position, Geschwindigkeit)



- **2-dimensionale Ebene / Torus**



- Die Zuordnung von Daten zu Knoten ist auf beliebig viele Dimensionen erweiterbar.
  z. B. 3D-Torus für Strömungssimulation

7.2.1.3. Message Passing Interface

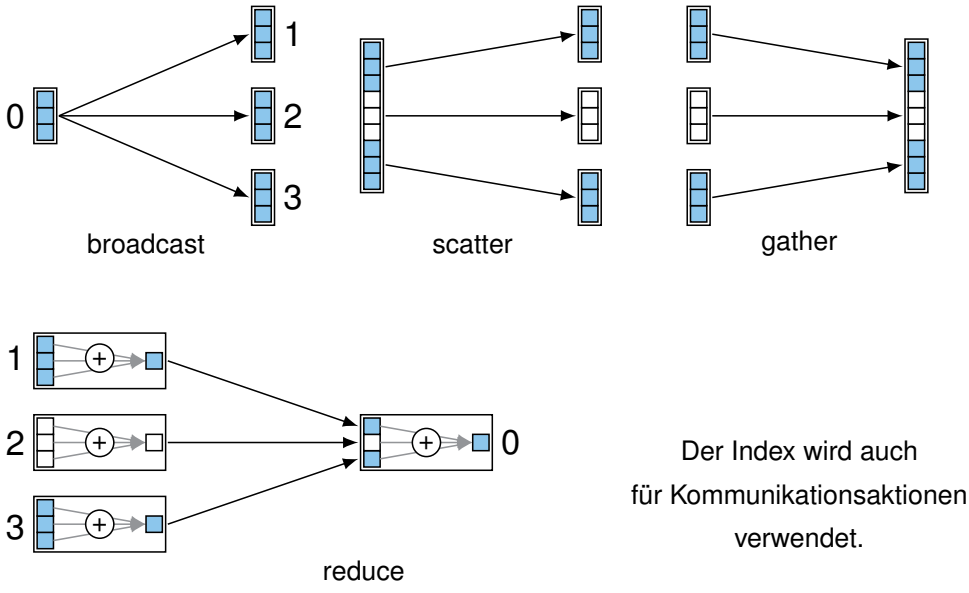
- Das Message Passing Interface (MPI) ist Standard auf Super-Computern und Compute Clustern.
- MPI hat ein standardisiertes Interface, das es erlaubt, Anwendungen auf unterschiedliche MPI-Implementierungen zu portieren.
- Die bekanntesten, freien Implementierungen sind:
  - Open MPI (`http://www.open-mpi.org/`)
  - MPICH2 (`http://www.mcs.anl.gov/research/projects/mpich2/`)
- Üblicherweise wird MPI mit C oder Fortran verwendet.
- MPI bietet verschiedene Abstraktionen von der reinen Socket-Programmierung:
  - Nachrichtenorientierte Kommunikation – MPI übernimmt den Aufbau von (TCP-)Verbindungen. Nachrichten können eine (fast) beliebige Länge haben.
  - MPI bietet Primitive für häufige Kommunikationsmuster wie z. B. "1 zu N" (`MPI_BCast`).
  - Man kann MPI Knoten in virtuellen k-dimensionalen kartesischen Räumen oder Tori anordnen lassen. Auch hierfür werden Primitive für die einfache Kommunikation angeboten.

7.2.1.4. Code-Beispiel: MPI

- Alle N MPI-Prozesse werden von 0 bis $N-1$ durchnummeriert.
- Aus seinem Index kann kann ein Prozess berechnen, für welche Teile der Daten er zuständig ist und wie er sich verhalten soll (analog zur Process-ID bei einem „fork“).
- Alle Knoten führen dasselbe Programm aus. Das Laufzeitsystem bestimmt für jede Instanz den Index.

```
#include <stdio.h>
#include <mpi.h>
int rank;  // enthaelt spaeter Index dieses Prozesses
int size;  // enthaelt spaeter Anzahl aller Prozesse
int main(int argc, char **argv) {
        MPI_Init(&argc, &argv);  // MPI-Laufzeitsystem sta
        MPI_Comm_rank(MPI_COMM_WORLD, &rank);  // Index ab
        MPI_Comm_size(MPI_COMM_WORLD, &size);  // Gesamtzal
        printf("I␣am␣%␣d␣of␣%d.\n", rank, size);
        MPI_Finalize();
}
```

7.2.1.5. Beispiele für Kommunikationsmuster



7.2.1.6. Code-Beispiel: MPI

Deklaration

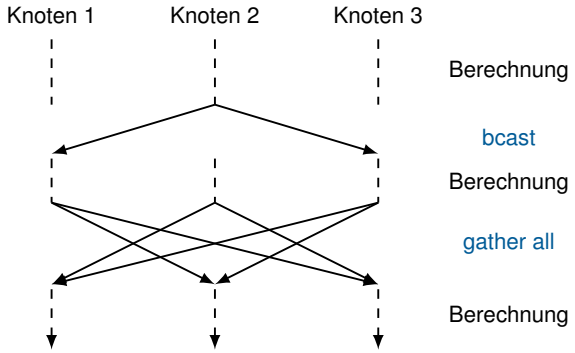
```
int MPI_Bcast(void *buffer, int count, MPI_Datatype datatyp
```

Beispiel-Aufruf

```
MPI_Bcast(in, inCount, MPI_INT, MASTER, MPI_COMM_WORLD);
```

- `MPI_Bcast` kopiert `count` Elemente des Typs `datatype` aus dem `buffer` des Prozesses mit dem Index `root` zu allen anderen. Nach dem Aufruf enthalten alle `buffer` aller Prozesse dieselben Daten.
- datatype: MPI hat eigene Datentypen, z. B. MPI_INT
  → Plattformunabhängigkeit!
- root ist der Index des Prozesses, der die Daten verteilt.
- comm: Knoten können in Kommunikationsdomänen unterteilt werden. Nur Knoten, die in der angegebenen Domäne sind, führen diesen Kommunikationsschritt aus.
  Alle Rechner befinden sich automatisch in `MPI_COMM_WORLD`.

7.2.1.7. Kosten



- In jedem Kommunikationsschritt muss auf den langsamsten Knoten gewartet werden.
- Die gesamten Kommunikationskosten sind die Summe der Kosten der einzelnen Kommunikationsschritte.
- Die Gesamtkosten können sich erheblich verteuern, wenn die Latenz im Netz groß ist.

7.2.1.8. Probleme

- Kommunikationsaktionen überlappen sich nicht, sondern blockieren das Programm → Performance-Verlust:
  - Amdahls Gesetz:
    $$\text{Speedup } s = \frac{1}{(1-p) + p/n}$$
  - Bei $n = 100$ Rechnern und einem parallelen Anteil von $p = 90\,\%$ beträgt der Speedup $s = 9.2$, also nur $9.2\,\%$ des Optimums von $s = 100$
- Die Topologie der Rechner und die Aufteilung der Daten sollten gut zueinander passen: “Verschnitt” führt zu schlechter Auslastung
- MPI leistet kein Load-Balancing: Ein einzelner Rechner, der mehr zu berechnen hat oder mit anderen Tasks ausgelastet ist, bremst alle anderen MPI-Prozesse aus.
- MPI beinhaltet prinzipbedingt nur sehr begrenzte Fehlertoleranz. Der Ausfall eines Rechners führt zu:

- Abbruch des ganzen Programms
- ... oder ein anderer Knoten übernimmt die Aufgabe.

  Im ungünstigsten Fall bedeutet doppelte Arbeit für einen Knoten eine Halbierung der Gesamt-Performance!

7.2.1.9. Feature-Tabelle

	MPI	**Anmerkungen**
reguläre Datenstrukturen	×	
skaliert automatisch	×	

- Der Programmierer bestimmt explizit, wie die Daten auf die Rechner aufgeteilt werden. Mit dem entsprechenden Vorwissen kann so für reguläre Datenstrukturen der Durchsatz maximiert werden.
- MPI ist ungeeignet für „tiefe“, irreguläre Datenstrukturen, wie z. B. Bäume.

7.2.2. MapReduce

7.2.2.1. MapReduce

- Manche Probleme lassen sich als eine Abbildung (Map) gefolgt von einer Reduktion (Reduce) formulieren. Beispiele:
  - Wörter in Dokumenten zählen
  - Sortieren
  - Index-Invertierung
- MapReduce
  - erwartet vom Programmierer nur die Implementierung der sequentiellen Funktionen Map und Reduce.
  - übernimmt Parallelisierung, Last-Verteilung und Fehlertoleranz.
- Beispiel: Wir wollen für jedes Assignment zählen, wieviele Teams mindestens einen Punkt erreicht haben.
  - Map

    Eingabe die `grades.txt` eines Teams, z. B. `team000/grades.txt`

    Ausgabe eine Liste aus Paaren der Form:
    `(Assignment-Nummer, erfolgreich)`
    `erfolgreich` ist 0 oder 1, wenn Zahl der Punkte ≥ 1
  - Reduce

    Eingabe ein Paar der Form:
    `(Assignment-Nummer, Liste aus ,,erfolgreich‘‘)`

    Ausgabe ein Paar der Form:
    `(Assignment-Nummer, Summe der Liste)`

7.2.2.2. Funktionsweise von MapReduce

- Die Funktionen „map“ (Abbildung) und „reduce“ (Reduktion[1]) stammen aus der funktionalen Programmierung und haben üblicherweise folgende Signaturen:

$$\begin{aligned} \text{map}: &\quad (\alpha_1 \longrightarrow \alpha_2) \times (\alpha_1\,\text{list}) \longrightarrow (\alpha_2\,\text{list}) \\ \text{reduce}: &\quad (\beta \times \beta \longrightarrow \beta) \times (\beta\,\text{list}) \longrightarrow \beta \end{aligned}$$

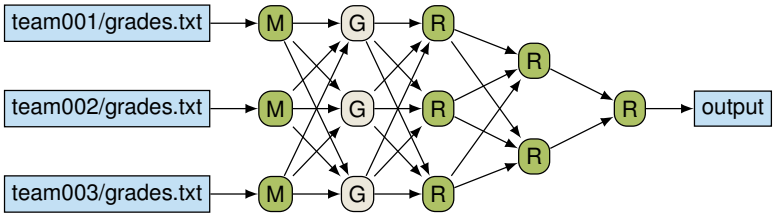
- Bei MapReduce entsprechen die Funktionen “Map” und “Reduce” dem ersten Parameter in obigen Signaturen – allerdings sind die Typen spezialisiert auf Key-Value-Paare:

$$\begin{aligned} \text{Map}: K_1 \times V_1 \longrightarrow &\quad (K_2 \times V_2)\,\text{list} \\ \text{group}: &\quad (K_2 \times V_2)\,\text{list} \longrightarrow (K_2 \times (V_2\,\text{list}))\,\text{list} \\ \text{Reduce}: &\quad K_2 \times (V_2\,\text{list}) \longrightarrow V_2\,\text{list} \end{aligned}$$

  wobei $\alpha_1 \mathrel{\widehat{=}} K_1 \times V_1$, $\alpha_2 \mathrel{\widehat{=}} (K_2 \times V_2)\,\text{list}$, ...

- Zwischen der „Map“- und der „Reduce“-Phase gruppiert das Framework die Werte nach Schlüsseln (group). Diese Funktion wird nicht vom Programmierer implementiert.

7.2.2.3. Datenfluss



- Für jede Datei wird eine Map-Operation (M) instanziiert. Diese Aufrufe sind völlig unabhängig voneinander und können parallel ausgeführt werden.
- Die beiden Reduktionsoperationen, Group (G) und Reduce (R), können bereits anlaufen, sobald Map genügend Ausgabe-Paare produziert hat.

  Die Ausführung der Phasen kann sich also überlappen, sodass die Rechner am Anfang nur Map und erst am Schluss nur noch Reduce-Operationen ausführen.
- Ein (fehlbarer) Master erzeugt die Arbeitspakete und verteilt sie erneut, wenn ein Slave abstürzt.

7.2.2.4. Code-Beispiel: MapReduce

- Auszählen bestandener Assignments in C++

```
virtual void Map(const MapInput& input) {
        const string& grades = input.value();
        // grades parsen, fuer jedes assignment und passed zu
        Emit(assignment, Emit(IntToString(passed));
}
```

[1] Die „reduce“-Funktion wird manchmal auch „fold“ genannt.

```
virtual void Reduce(ReduceInput* input) {
        int64 sum = 0;
        while (!input->done())  {
                sum += StringToInt(input->value());
                input->NextValue();
        }
        Emit(IntToString(sum));
}
```

7.2.2.5. MapReduce

- MapReduce wurde von Google – ursprünglich für den internen Gebrauch – entwickelt [3].
- Apache Hadoop MapReduce ist eine freie Implementierung von MapReduce.
- Nachteile
  - MapReduce ist nur für Probleme geeignet, die sich in genau eine Abbildung und/oder eine Reduktion zerlegen lassen – MapReduce kennt keine anderen Kommunikationsmuster.
  - Der Programmierer hat keinen Einfluss darauf, wie die Tasks im System verteilt werden. Das könnte zu schlechterer Performance führen, wenn der Programmierer dadurch a priori-Wissen über die Struktur des Systems nicht in die Lösung einbringen kann.

7.2.2.6. Feature-Tabelle

	MPI	MapReduce
reguläre Datenstrukturen	×	
skaliert automatisch	×	×
automatische Verteilung		×
Fehlertoleranz		×

7.2.3. Pipes

7.2.3.1. Pipes

Definition (Pipe):

Ein Simplex-Datenstrom, der durch ein File Handle repräsentiert und durch das Betriebssystem gepuffert wird, nennt man Pipe.

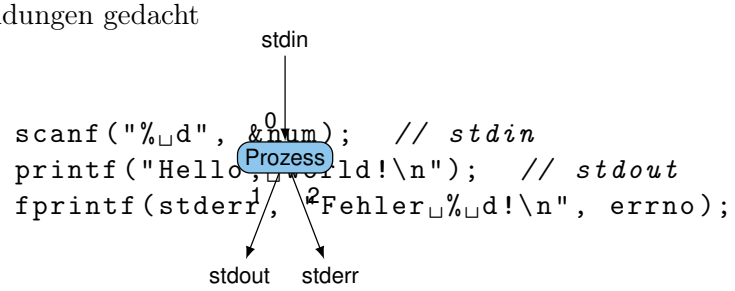
- Pipes ermöglichen Inter-Prozess-Kommunikation (IPC).
- Jeder Prozess verfügt (nach dem POSIX-Standard) automatisch über drei Pipes mit vorgegebenen Nummern (File Handles):

stdin 0, Eingabe, oft Tastatur oder Datei

stdout 1, Ausgabe, normaler Ausgabedatenstrom

stderr 2, Ausgabe, für Fehlermeldungen gedacht

- Zugriff in C (Beispiele):

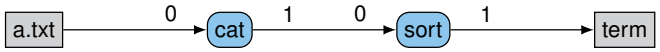



```
scanf("␣d", &num);   // stdin
printf("Hello,␣World!\n");   // stdout
fprintf(stderr, "Fehler␣%␣d!\n", errno);
```

7.2.3.2. Pipes umleiten mit sh und netcat

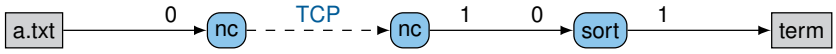
- Beispiel 1: Die Datei `a.txt` soll vom Programm `cat` eingelesen und der Inhalt an `sort` zum Sortieren weitergereicht werden (IPC):

```
host1> cat a.txt | sort
```



- Beispiel 2: Der Dateiinhalt soll auf `host1` gelesen und auf `host2` sortiert werden. Dazu können die Daten zum Beispiel via netcat oder SSH auf einen anderen Rechner übertragen werden:

```
host2> nc -lp 1234 | sort
host1> nc host2 1234 <a.txt
```



- Anmerkung: netcat verschlüsselt den Datenstrom nicht. Für unsichere Netzwerke wie Funkverbindungen sollte SSH verwendet werden.

- Problem: Bei einem komplexeren Prozess-Graph müssen zuerst die Pipes angelegt werden und dann auf möglicherweise vielen Rechnern die nötigen Prozesse gestartet werden. Das ist umständlich.

7.2.3.3. DUP

- Das DUP-System [4] dient dem einfachen Aufbau eines Graphen aus Prozessen, die über Pipes miteinander verbunden sind.

- Der DUP-Dämon (`dupd`) wird einmal pro teilnehmendem Rechner gestartet und wartet dann auf eingehende TCP-Verbindungen. Er übernimmt folgende Aufgaben:
  - Pipes erstellen
  - Prozesse mit Pipes verbinden und starten

- Der Prozess-Graph wird mit einer einfachen, domänenspezifischen Sprache beschrieben.

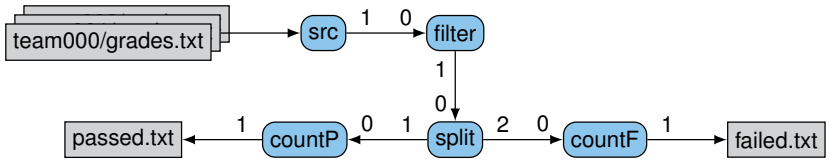
- DUP-Syntax (vereinfacht):
  Name `@` Host `[` Umleitungen `]` `$` Shell-Befehl `;`

- Umleitungen beschreiben die Struktur des Graphen als Adjazenzliste; Beispiele für Umleitungen:
  - `0 < foo.txt` – Inhalt der Datei "foo.txt" nach stdin kopieren
  - `1 | foo:0` – stdout zu stdin vom Prozess mit dem Namen "foo" umleiten
  - `2 >> errors.log` – stderr in die Datei "errors.log" umleiten

7.2.3.4. Code-Beispiel: DUP

- Wieviele Gruppen haben in Assignment 1 volle Punktzahl?

Datenfluss



Quell-Code

```
src@192.168.1.1 [ 1|filter:0 ] $ cat team*/grades.txt;
filter@192.168.1.2 [ 1|split:0 ] $ grep 'assignment01:';
split@192.168.1.3 [ 1|countP:0, 2>countF:0 ] $ mgrep '01: 2';
countP@192.168.1.4 [ 1>passed.txt ] $ wc -l;
countF@192.168.1.5 [ 1>failed.txt ] $ wc -l;
```

7.2.3.5. Feature-Tabelle

	MPI	MapReduce	DUP	
reguläre Datenstrukturen	×		×	**Anmerkungen**
skaliert automatisch	×	×		
automatische Verteilung		×		
Fehlertoleranz		×		
heterogen			×	

- Der Prozess-Graph in einem DUP-Skript ist statisch und passt sich weder an die Größe des Systems noch an die Datenmenge an.

- DUP ermöglicht es, bestehende Konsolenprogramme in den Prozess-Graphen zu integrieren.

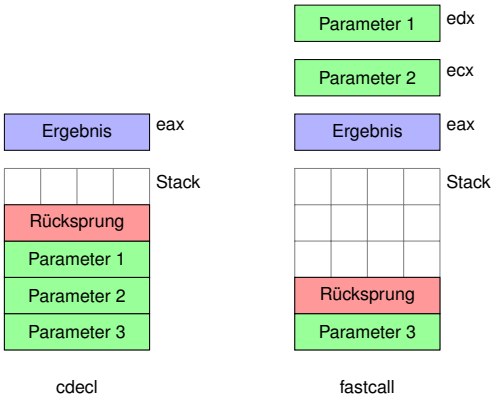
7.3. Remote Procedure Call

7.3.0.6. Remote Procedure Call

- Die meisten Programmiersprachen unterstützen Funktionen (oder Prozeduren), um Teile des Quell-Codes zu abstrahieren.
- Eine Funktion isoliert den inneren Code vom äußeren und hat ein klar definiertes Interface. Funktionen bieten sich daher für die Kommunikation zwischen Komponenten auf verschiedenen Rechnern an.
- Ein Remote Procedure Call (RPC) ist ein Funktionsaufruf über Knotengrenzen hinweg.
- Dabei liegt das Client-/Server-Modell zugrunde:
  Der Server bietet eine Funktion an, der Client ruft sie auf.

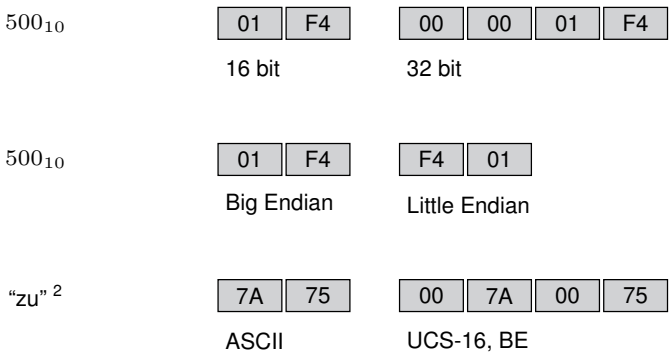
7.3.0.7. Was ist ein Funktionsaufruf?

- Sprünge sind nur innerhalb des eigenen Adressraums möglich.
- Daten sind i. A. ebenfalls auf den eigenen Adressraum beschränkt.
  Beispiel: Zwei Aufrufkonventionen für die IA32-Architektur (x86):



- Es gibt mehrere, inkompatible Aufrufkonventionen (calling conventions), die bestimmen, wie die Parameter übergeben werden. Üblicherweise werden die Daten über den Stack und/oder Register übergeben.

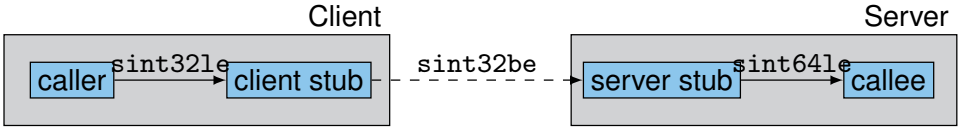
7.3.0.8. Wie sind die Parameter codiert?



- Fazit: Eine Funktion in einem anderen Prozess kann i. A. nicht direkt aufgerufen werden.

7.3.0.9. Stubs

- Für jede Funktion, die per RPC aufrufbar sein soll, wird sowohl für die Server- als auch die Client-Seite ein Stub erzeugt.
- Ein Stub ist eine Funktion, die anstelle der eigentlichen Funktion aufgerufen wird. Der Stub ...
  1. konvertiert die Parameter der Remote Procedure in eine gemeinsame Darstellung und packt sie in eine Nachricht (Marshalling).
  2. verschickt die Nachricht an den Server.
  3. blockiert die Programmausführung, bis eine Antwort eintrifft.
  4. konvertiert die Daten der Antwort zurück in die interne Darstellung (Unmarshalling).
- Auf der Server-Seite befindet sich ebenfalls ein Stub, der die Parameter in die Server-interne Darstellung konvertiert, die gewünschte Funktion aufruft und das Ergebnis zurücksendet.
- Bei entsprechender Unterstützung durch die Programmiersprache (z. B. RMI in Java) ist der Aufruf transparent und nicht von einem lokalen Aufruf unterscheidbar (Ausnahme: Verbindungsfehler).



Darstellung eines Funktionsaufrufs (Pfeile) über Knotengrenzen hinweg.

7.3.0.10. IDL

- Stubs müssen nicht von Hand implementiert werden, sondern können automatisch für verschiedene Programmiersprachen erzeugt werden.
- Die Interface Description Language (IDL) definiert Funktionen und Datentypen weitgehend sprachunabhängig. Aus IDL-Definitionen werden die Stubs generiert.

Beispiel:

IDL `long min([in] long a, [in] long b);`

C `long min(long a, long b);`

Java `int min(int a, int b);`

- Die Implementierung der Netzwerkkommunikation und der verwendeten Protokolle werden normalerweise in einer Bibliothek gekapselt, gegen die das Anwendungsprogramm gelinkt wird. Folge:
  - Der Anwendungsprogrammierer benötigt keine Kenntnisse über die verwendeten Protokolle.
  - Mit Hilfe derselben Spezifikation können Stubs für verschiedene Protokolle implementiert werden.

7.3.0.11. Binding

- Da der Client-Stub zur Kompilierungszeit noch nicht die Transport-Adresse des Servers kennt (Dynamic Binding[2]), fragt er zur Laufzeit einen dedizierten Dienst, z. B. rpcbind.
- rpcbind erwartet Anfragen normalerweise auf Port 111.
- Die Server-Prozesse registrieren sich bei rpcbind und teilen ihm folgende Informationen mit:
  - Name des Server-Prozesses
  - Programmnummer und Versionsnummer
  - IP-Adresse und Port unter der der Server erreichbar ist

7.3.0.12. Java RMI

- Java ist eine objektorientierte Sprache mit komplexen Datentypen, Referenzen und Garbage Collection.
- Java Remote Method Invocation (RMI) ist eine RPC-Variante, die auf Java zugeschnitten ist. RMI ermöglicht es, komplexe Objekte und Objektgraphen als Parameter zu übergeben und unterstützt *verteilte Garbage Collection*.
- Entfernte Objekte bleiben referenziert und werden nicht kopiert. Anstelle von einzelnen Stubs, also Funktionen, erfüllen bei RMI auf Client-Seite ein Proxy-Objekt und auf Server-Seite ein Skeleton-Objekt diese Rolle.

7.3.0.13. SOAP

- Die bisher betrachteten Stubs und Proxies verwenden ein eigenes, binäres Format. Das jüngere Simple Object Access Protocol (SOAP) baut dagegen auf bestehenden WWW-Standards auf.
  - Das Nachrichtenformat basiert auf XML Information Set (XML Infoset) und wird normalweise als XML serialisiert[3].
  - Eine SOAP-Nachricht besteht aus einem optionalen Header mit Adressierungsinformationen und einem Body, der mehrere Nachrichten enthalten kann.

[2] vgl. "Dynamic Linking" aus der Vorlesung zu Betriebssystemen
[3] Andere Formate wie JSON sind aber ebenfalls möglich.

- SOAP spezifiziert lediglich das Nachrichtenformat, nicht das Transportprotokoll. SOAP arbeitet nicht verbindungsorientiert, sondern nachrichtenorientiert. In der Regel wird für den Transport HTTP verwendet.
- Im Gegensatz zu RPC kann man mit SOAP auch Nachrichten spezifizieren, die keine Antwort erfordern.
- Analog zu Internet Mail kann eine SOAP-Nachricht von einem Sender („originator“) über mehrere Zwischenknoten („intermediary“) an einen Empfänger weitergeleitet werden. Die Zwischenknoten interpretieren nur den Header.

7.3.0.14. Beispiel einer SOAP-Nachricht (XML)

Beispiel

```
<?xml version="1.0" ?>
<soap:Envelope xmlns:soap="http://www.w3.org/2003/05/soap-
<soap:Header>
<t:TransactionCode soap:mustUnderstand="1" xmlns:t="http:/
</soap:Header>
<soap:Body>
<m:GetPost xmlns:m="http://www.example.com/blog">
<m:PostID>123</m:PostID>
</m:GetEntries>
</soap:Body>
</soap:Envelope>
```

Aufbau

- 2–13: Der Envelope einer SOAP-Nachricht ist der Wurzelknoten des XML-Dokuments.
- 3–5: Der Header ist optional und kann leer sein. `mustUnderstand` bedeutet, dass Empfänger dieses Feld nicht ignorieren dürfen.
- 8–12: Der Body enthält die Parameter des Aufrufs. SOAP schreibt kein bestimmtes Format des Bodys vor, sondern überlässt anderen Standards, z. B. WSDL, die Spezifikation für bestimmte Protokolle.

In Java: So könnte die Signatur einer zugehörigen Java-Methode aussehen:
`List<Post> GetPost(int PostID);`

7.3.0.15. WSDL

Ein angebotener Dienst kann mit Hilfe der Web Services Description Language (WSDL) beschrieben werden. Eine WSDL-Beschreibung wird durch ein XML-Dokument repräsentiert, das folgende Informationen enthält:

Abstrakte Beschreibung

- Komplexe Typen können aus vordefinierten, primitiven Typen zusammengesetzt werden.

- Nachrichten definieren mit Hilfe von Typen das Format der Daten, die ausgetauscht werden.
- Operationen, welche durch eine Sequenz an Nachrichten beschrieben werden.
- Endpunkt-Typen beschreiben die Operationen, die ein Kommunikationspartner unterstützt.

Konkrete Beschreibung

- Ein Binding spezifiziert für einen Endpunkt-Typ ein konkretes Protokoll und Nachrichtenformat (z. B. SOAP über HTTP).
- Ein Endpunkt ist eine Kombination aus einem Binding und einer Netzwerkadresse.
- Ein Dienst ist eine Menge an zusammengehörigen Endpunkten.

Im Gegensatz zur IDL kann eine WSDL-Beschreibung auch die konkrete Adresse eines Dienstes enthalten. Auch aus WSDL-Beschreibungen kann automatisch Quell-Code generiert werden, der Netzwerkkommunikation und Marshalling übernimmt.

7.3.0.16. Web Services

Die Architektur von Web Services wird vom W3C spezifiziert und kann als RPC über das Internet auf Basis von WWW-Technologien betrachtet werden.

Architektur (stark vereinfachte Darstellung)

- **Transport**: Die Transportschicht dient der Übertragung von XML-Nachrichten.
  Implementierungen: Normalerweise HTTP, aber auch: SMTP, FTP, ...
- **Messaging**: Definiert das Rahmenformat für (XML-)Nachrichten.
  Implementierungen: Normalerweise SOAP, aber auch XML-RPC
- **Description**: Die Dienstbeschreibung definiert analog zu *IDL* das Interface eines Web Services: Funktionen und ihre Parameter, wo und über welches Protokoll der Dienst angeboten wird.

  Implementierungen: WSDL
- **Discovery**: Die Dienstschicht erfüllt Funktionen analog zu *rpcbind*, auf einer höhereren Ebene: Registrierung und Lokalisation von Diensten und Beschreibung ihrer Interfaces.

  Implementierungen: Universal Description Discovery and Integration (UDDI)

Prinzipielle Unterschiede zu RPC

- Typischerweise wird RPC nur in lokalen Netzen verwendet. Web Services können dagegen über das Internet verteilt sein.
- Web Services bieten höhere Abstraktionsschichten, die über die rein technischen Aspekte des Nachrichtentransports hinausgehen. Deshalb sind die Kommunikationspartner im Vergleich weniger eng an bestimmte Technologien gebunden.

7.3.0.17. „Web Services“ in the Wild

- Die „höheren“ Schichten, insbesondere UDDI, haben sich in der Praxis nicht durchgesetzt.
- Die „niedrigeren Schichten“ von Web Services, insbesondere der Nachrichtenversand über HTTP, sind im breiten Einsatz.
- Anstelle von SOAP und WSDL werden oft frei-definierte Nachrichten im XML- oder JSON-Format an einen Web Server übertragen. Der Begriff „XML-RPC“ bezeichnet oft nicht das ursprüngliche, gleichnamige Protokoll, sondern allgemein die Übertragung beliebig spezifizierter XML-Nachrichten über HTTP.
- Representational State Transfer (REST) ist ein verbreitetes Architekturmodell, das vorsieht, das der Anbieter eines Dienstes zwischen Nachrichten keine verbindungsspezifischen Daten halten muss. Dies ermöglicht hohe Skalierbarkeit auf der Server-Seite.

7.3.0.18. RPC

Vorteile

- Dank einer eindeutigen Spezifikation und daraus generiertem Code können heterogene verteilte Systeme konstruiert werden, die aus unterschiedlichen
  - Hardware-Plattformen,
  - Betriebssystemen und
  - Programmiersprachen

  bestehen.

Nachteile

- Marshalling kann durch den Serialisierungsaufwand teuer sein, weil die Daten vor dem Versand konvertiert und zum Teil kopiert werden müssen.
- RPC ist ein Low-Level-Konstrukt, das weder Fehlertoleranz noch automatische Lastverteilung bietet.

7.3.0.19. Feature-Tabelle

	MPI	MapReduce	DUP	RPC/RMI
reguläre Datenstrukturen	×		×	×
skaliert automatisch	×	×		
automatische Verteilung		×		
Fehlertoleranz		×		
heterogen			×	×
Marshalling	·			×

7.4. Shared Memory

7.4.0.20. Shared Memory

Definition:

Shared Memory ist Arbeitsspeicher, der von mehreren Prozessen gleichzeitig verwendet werden kann.

- Innerhalb eines sequentiellen Programms können Programmteile über Variablen und Objekte miteinander kommunizieren.
- Threads, die demselben Prozess angehören, arbeiten im gleichen Adressraum und können daher ebenfalls über den Arbeitsspeicher miteinander kommunizieren.
- Wenn Shared Memory zur Verfügung steht, kann eine verteilte Anwendung wie eine Multi-Threaded-Anwendung geschrieben werden. Dadurch erspart man sich das manuelle Kopieren von Daten zwischen den Prozessen via Message-Passing.
- Für die Performance ist aber auch in einem Shared Memory-System wichtig, wie die Daten auf die Rechner verteilt werden.

7.4.1. NUMA

7.4.1.1. NUMA

- Mit NUMA-Architektur (Non-Uniform Memory Access) bezeichnet man einen Cluster, dessen Netzwerk den Prozessoren erlaubt, direkt auf die Speicher der anderen Prozessoren zuzugreifen.
- Die einzelnen Arbeitsspeicher der Rechner erscheinen dem Betriebssystem dabei wie ein einzelner, großer Arbeitsspeicher (Single System Image).

Vorteile

- Zur Parallelisierung muss ein Programm lediglich so angepasst werden, dass es mehrere Prozessorkerne nutzen kann.
- Da es sich um eine Hardware-Lösung handelt, ist die Latenz beim Zugriff auf entfernten Speicher geringer als bei Software-Lösungen.

Nachteile

- "Non-Uniform" bezieht sich darauf, dass ein Prozessor aufgrund der Netzwerklatenz auf Adressen, die in den eigenen Arbeitsspeicher anstatt in entfernten verweisen, weitaus schneller zugreifen kann. Folglich müssen die Daten trotz der einheitlichen Sicht auf den Speicher sorgfältig im Adressraum angeordnet werden.
- Die Caches aller Prozessoren kohärent zu halten skaliert schlecht. Ohne Cache-Kohärenz können die Programme wesentlich komplexer werden.

7.4.2. Paging

7.4.2.1. Virtueller Speicher

- Moderne Prozessoren verfügen über eine Memory Management Unit (MMU), die in Kooperation mit einem geeigneten Betriebssystem den Arbeitsspeicher des Systems virtualisieren kann.

  Beispiele: Linux, BSD, Windows (ab NT), MacOS (ab OS X)

- Der physische Arbeitsspeicher wird in Kacheln unterteilt, die typischerweise 4 KiB groß sind. Jedem Prozess wird ein eigener Adressraum zugeordnet, der in Seiten der gleichen Größe untergeteilt wird.

- Die Seitentabelle bildet für jeden Prozess die benutzten Seiten auf Kacheln ab. Die Seitentabelle entpricht dabei einer Abbildung:

  $\text{Seitentabelle} : \text{Prozesse} \times \text{Seiten} \to \text{Kacheln}$

- Die MMU verwendet bei jedem Speicherzugriff eines Prozesses die Seitentabelle, um die virtuelle Adresse des Prozesses in eine physische Adresse zu übersetzen.

- Dieser Vorgang ist für die Prozesse völlig transparent. Jeder Prozess kann nur auf seinen eigenen Adressraum zugreifen.

7.4.2.2. Distributed Shared Memory über Paging

Auslagern

- Sollte in der Seitentabelle zu einer virtuellen Adresse kein gültiger Eintrag vorhanden sein, wird das Betriebssystem benachrichtigt (Interrupt Request). Das Betriebssystem hat dann die Möglichkeit, die Seite auf eine gültige Kachel abzubilden und den Prozess fortzusetzen.

- Diesen Umstand nutzt das Betriebssystem, um Seiten in einen Massenspeicher auszulagern und bei Bedarf zurückzuholen.

Distributed Shared Memory

- *Ansatz:* Anstatt zwischen Hauptspeicher und Massenspeicher werden die Seiten zwischen den Hauptspeichern eines Clusters über das Netzwerk getauscht.

- Der Adressraum eines Prozesses kann sich so über die Kacheln vieler Rechner erstrecken. Die Rechner erscheinen dabei wie ein einzelner Rechner mit vielen Rechenkernen[4]. Der lokale Arbeitsspeicher verhält sich dabei wie ein Cache im Prozessor.

- [5] vergleicht verschiedene Methoden, den Zugriff auf und Austausch von Seiten zu verwalten.

[4] Analog zu Simultaneous Multithreading (SMT) heutiger Multi-Core-Prozessoren.

7.4.2.3. Probleme

- Kacheln sind typischerweise 4 KiB groß. Diese Größe wird durch die Hardware vorgeschrieben. Die meisten Variablen und Objekte sind kleiner. Folge: Sehr verschiedene Daten können in derselben Seite liegen.
- Über das Speichersystem können nur ganze Seiten angefordert werden. Wenn zwei verschiedene Prozessoren ständig auf zwei verschiedene Variablen zugreifen, die in derselben Seite liegen, muss diese Seite ständig zwischen den beiden Prozessoren hin- und hergeschoben werden. Dies ist von der Programmlogik her unnötig, weil die Prozessoren auf unterschiedlichen Daten arbeiten und daher voneinander unabhängig arbeiten könnten. Dieses Problem bezeichnet man als False Sharing.
- Unter anderem aufgrund der Performance-Probleme durch False Sharing findet dieser Ansatz in der Praxis nur im High-Performance-Computing Anwendung. Die regulären Datenstrukturen, die hier typischerweise verarbeitet werden, erlauben eine sorgfältige Platzierung der Daten (vgl. MPI).

7.4.3. Sharing auf Objektebene

7.4.3.1. Konsistenz in parallelen Programmen

- Die Funktion `remove` löscht ein Element aus einer doppelt-verketteten Liste:

```
void remove(list_element *el) {
        el->prev->next = el->next;
        el->next->prev = el->prev;
        free(el);
}
```

- *Problem:* Nebeneinanderliegende Elemente können gleichzeitig von verschiedenen Threads gelöscht werden. Dabei befindet sich die Liste zwischen Zeile 2 und Zeile 3 in einem Zustand, der für andere Threads inkonsistent ist, also invarianten der Datenstruktur verletzt.
- Traditionell wird dieses Problem behoben, indem sich ein Thread mit Hilfe von Locks exklusiven Zugriff auf die benötigten Daten sichert.

7.4.3.2. Code-Beispiel: Locking

- Thread-sicheres Entfernen eines Elements aus einer Linked List mit Hilfe von Locks

```
void remove(list_element *el) {
        acquire(el->prev->lock);
        acquire(el->lock);
        acquire(el->next->lock);
        el->prev->next = el->next;
        el->next->prev = el->prev;
```

```
        release(el->prev->lock);
        release(el->lock);
        release(el->next->lock);
        free(el);
}
```

7.4.3.3. Code-Beispiel: Locking

- Achtung: Diese Variante führt zu Dead-Locks! Beispiel: Das aktuelle Element und sein Vorgänger werden gleichzeitig gelöscht. Beide sperren sich zunächst selbst. Am Ende wartet der Vorgänger auf dieses Element (Z. 4) und dieses Element auf den Vorgänger (Z. 3).

```
void remove(list_element *el) {
        acquire(el->lock);
        acquire(el->prev->lock);
        acquire(el->next->lock);
        el->prev->next = el->next;
        el->next->prev = el->prev;
        release(el->lock);
        release(el->prev->lock);
        release(el->next->lock);
        free(el);
}
```

7.4.3.4. Software Transactional Memory

- Das Problem mit der Funktion `remove` ist, dass inkonsistenter Zwischenzustand beobachtet werden kann, während die Funktion ausgeführt wird, weil die Operationen nacheinander aus dem Speicher lesen und darauf schreiben.
- Das Problem würde nicht bestehen, wenn die Funktion atomar wäre, d. h., wenn sie gleichzeitig alle ihre Eingaben lesen und alle Ausgaben schreiben könnte.
- Software Transactional Memory bietet die Möglichkeit, einen Block aus Anweisungen als Transaktion zu markieren.
- Transaktionen sind atomar. Sie werden entweder ganz oder gar nicht ausgeführt und verhalten sich so, als ob sie während ihrer Ausführung eine Momentaufnahme des Gesamtsystems gesehen hätten.

7.4.3.5. Software Transactional Memory

- Der Code innerhalb des atomic Blocks wird in einer Transaktion ausgeführt:

```
                        void remove(list_element *el) {
                                atomic {
// begin transaction
                                        el->prev->next = el->next;
```

```
                                        el->next->prev = el->prev;
                                    }
    // commit transaction
                                    free(el);
                                }
```

- Am Ende betritt eine Transaktion die Commit-Phase. Hier überprüft das System, ob während der Ausführung der Transaktion andere Transaktionen abgeschlossen wurden, die in Konflikt stehen.

- Ein Konflikt entsteht z. B. dann, wenn zwei laufende Transaktionen auf die gleiche Variable schreiben. In diesem Fall wird eine der beiden Transaktionen zurückgerollt und wiederholt.

7.4.3.6. Veranschaulichung

Umgangssprachlich ausgedrückt kann man den Zwischenzustand einer Operation nicht beobachten, weil ...

Atomar: zwischen Anfang und Ende keine Zeit vergeht.

Locking: es verboten ist, hinzusehen.

STM: man sich hinterher darauf einigt, dass man es nicht gesehen hat.

7.4.3.7. Distributed Software Transactional Memory

- Verteilter STM benötigt eine zentrale Anlaufstelle oder ein Konsens-Protokoll, damit die beteiligten Prozesse im Konfliktfall entscheiden können, welche Transaktionen zurückgerollt werden müssen.

- Beispiel: DecentSTM (dezentrales verteiltes Konsensprotokoll)

Nachteile

- Innerhalb einer Transaktion dürfen außer dem Lesen und Schreiben von Variablen keine Operationen mit Nebenwirkungen durchgeführt werden, weil deren Wirkung nicht rückgängig gemacht werden kann.
  Beispiele: Auf die Konsole schreiben, Atomraketen starten, ...

- Um auf Konflikte prüfen zu können, müssen sich die beteiligten Prozesse abstimmen. Diese Aufgabe übernimmt
  - ein zentraler Server, der ein Single Point of Failure ist — oder
  - ein Konsens-Protokoll, das einen hohen Kommunikationsaufwand haben kann.

7.4.3.8. Feature-Tabelle

	MPI	MapReduce	DUP	RPC/RMI	NUMA	Seitenbasierter DSM	Distributed STM
reguläre Datenstrukturen	×		×	×	×		
skaliert automatisch	×	×			·	×	×
automatische Verteilung		×					·
Fehlertoleranz		×					·
heterogen			×	×			
Marshalling	·			×			

7.5. Einbettung in Programmiersprachen

7.5.0.9. Erlang

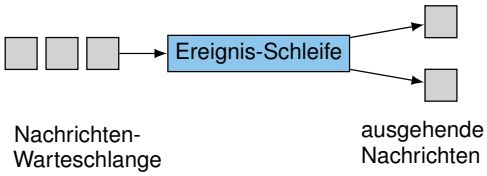
- Erlang ist eine dynamische, funktionale Programmiersprache mit primitiven Kommunikationsoperatoren.
- Ericsson hat inzwischen Erlang zusammen mit der verteilten In-Process-Datenbank „Mnesia“ und der „Open Telecommunication Platform“ (OTP) unter einer freien Lizenz veröffentlicht.
- Erlang wurde von Ericsson hauptsächlich entwickelt, um die Software für ihre ATM-Switches in einer Hochsprache implementieren zu können, die Kommunikation in einem verteilten System möglichst einfach macht.
- Bekannte Anwendungen außerhalb der Telecom-Branche sind:
  - ejabberd (Jabber-Server)
  - CouchDB (NoSQL-Datenbank)

Was ist an Erlang hier interessant?

- Erlang integriert das Actor Model direkt in die Programmiersprache.
- Erlang bietet verteilte Fehlerbehandlung.

7.5.0.10. Das Actor Model

- Ein Actor besteht aus einer Nachrichten-Warteschlange (event queue) und einem Prozess, der in einer Ereignisschleife (event handler) die Nachrichten abarbeitet.
- Actors identifizieren sich gegenseitig über Process Identifiers (PIDs) und können sich darüber gegenseitig Nachrichten zusenden. Da die PID auch den Rechner identifiziert, können Nachrichten über Rechnergrenzen hinweg gesendet werden.



- Explizite Synchronisationskonstrukte, wie z. B. Mutexes, sind nicht notwendig, weil die Nachrichtenwarteschlange ein natürlicher Synchronisationspunkt ist.
- Der Nachrichtenversand ist ungeordnet: Die Nachrichten treffen nicht notwendigerweise in der Reihenfolge in der Warteschlange ein, in der sie versandt wurden.

7.5.0.11. Erlang: Sequentielle Syntax

Primitive Datentypen

`a` Atom, ein Wert, der nur für sich selbst steht (wird meistens verwendet, um Nachrichten-Typen zu unterscheiden)

`{a, 23, "a"}` 3-Tupel, das aus einem Atom, einem Integer und einem String besteht

`foo(42)` Funktionsaufruf: Funktion "foo" mit Parameter 42

Definition einer Funktion

```
dist(X, Y) ->          %  Funktion 'dist' mit zwei Parametern
A = X*X + Y*Y,     %  lokale Variable (read-only binding)
sqrt(A).           %  Rueckgabewert ist Ergebnis von 'sqrt'
```

- Hinweis: Variablennamen beginnen in Erlang immer mit einem Großbuchstaben, Atome mit einem Kleinbuchstaben.

7.5.0.12. Erlang: Kommunikationsoperationen

Prozess starten

```
Pid = spawn(a, fun, args)     %  Neuen Prozess starten, der di
%    Funktion 'fun' ausfuehrt mit den
%    Parametern 'args' ausfuehrt
```

Nachricht senden

```
Pid ! Msg     %  Dem Prozess mit der PID 'Pid' den Wert 'Msg'
%    als Nachricht senden
```

Nachrichten empfangen

```
receive                                    %  Auf Nachricht war
hello          -> doSomething();         %  Mustervergleiche au
{"hi", Name} -> doOtherThing(Name)   %  ... empfangener Nachri
end.                                       %  ... (Fallunterschi
```

Anmerkung: Erlang hat eine eigenwillige Syntax mit verschiedenen Interpunktionszeichen (Komma, Semikolon, Punkt), um Befehle voneinander zu trennen. Das liegt daran, dass die Entwickler ursprünglich eine PROLOG-ähnliche Sprache implementieren wollten.

7.5.0.13. Code-Beispiel: Erlang

- Ein einfacher Echo-Server. Siehe http://www.erlang.org/doc/getting_started/users_guide.html für ausführlichere Beispiele.

```
echo_server() ->
receive                    %  Warte auf 'ping'-Nachricht.
{ping, Sender} ->
Sender ! pong,          %  Antworte mit 'pong'.
echo_server()           %  Rekursion, warte auf die naechste
end.                      %  ... Nachricht.
start_echo_server() -> spawn(demo, echo_server, []).
main() ->
Echo = start_echo_server(),     %  Variable 'Echo' enthaelt
Echo ! {ping, self()},          %  Sende 'ping' mit eigener
receive                         %  Warte auf Antwort.
pong -> io:format("Pong.~n", [])
end.
```

7.5.0.14. Programmieren in Erlang

- Das Starten neuer Prozesse ist in Erlang auch ein Mittel, um das Programm zu strukturieren. Das bedeutet, dass man auch ohne erwarteten Performancegewinn neue Prozesse startet. Dadurch stehen manchmal schon genügend Prozesse zur Parallelisierung zur Verfügung, wenn das Programm auf einem größeren System ausgeführt wird.

- Prozesse werden nicht automatisch verteilt. Über einen zusätzlichen Parameter teilt man `spawn` mit, auf welchem Rechner der Prozess gestartet werden soll.

7.5.0.15. Verteilte Fehlerbehandlung in Erlang (1/2)

- Fehler (signals) werden in Erlang wie spezielle Nachrichten der Form `{'EXIT', FromPid, Reason}` behandelt. Empfängt eine Warteschlange diese Nachricht, wird der Prozess sofort beendet.

- Der Befehl link verbindet zwei Prozesse so miteinander, dass Fehlernachrichten in beide Richtungen weitergeleitet werden und folglich auch die Beendigung des Partners auslösen. Diese Verbindung ist transitiv.

Zwei Prozesse miteinander verbinden (link)

```
start() ->
Pid = spawn(demo, some_proc, []),
link(Pid),
%  erledige andere Dinge
```

7.5.0.16. Verteilte Fehlerbehandlung in Erlang (2/2)

- Das Standard-Verhalten eines Prozesses bezüglich Signalen kann geändert werden:

```
process_flag(trap_exit, true)
```

- Dadurch wird die Fehlernachricht in eine gewöhnliche Nachricht umgewandelt, die der Prozess normal über die Warteschlange empfangen kann. Sie hat nicht mehr die Beendigung des Prozesses zur Folge und wird auch nicht mehr an verbundene Prozesse weitergeleitet.

```
receive
{'EXIT', FromPid, Reason} ->  %  ... Fehler behan
end.
```

Strategie:

- Alle Prozesse, die direkt voneinander abhängen, werden miteinander verbunden. Dadurch wird durch einen einzelnen Fehler eine Kaskade an Prozessen beendet.
- Nur solche Prozesse fangen Fehlernachrichten ab, die den Fehler sinnvoll behandeln können. Diese Prozesse starten dann rekursiv die beendeten Prozesse neu.

7.5.0.17. Feature-Tabelle

	MPI	MapReduce	DUP	RPC/RMI	NUMA	Seitenbasierter DSM	Distributed STM	Erlang
reguläre Datenstrukturen	×		×	×	×			×
skaliert automatisch	×	×			·	×	×	
automatische Verteilung		×					·	
Fehlertoleranz		×					·	×
heterogen			×	×				
Marshalling	·			×				·

A. Appendix

A.1. Binärpräfixe

Für physikalische Größen, beispielsweise Entfernung oder Spannungspegel, werden üblicherweise SI-Präfixe verwendet. Dabei handelt es sich um die umgangssprachlich auch für Datenmengen verwendeten Präfixe wie etwa „kilo“ oder „Mega“. Aus historischen Gründen werden Datenmengen aber häufig nicht als Vielfache von Zehner- sondern von Zweierpotenzen angegeben. Bei dem umgangssprachlichen „Kilobyte“ handelt es sich häufig um 1024 Byte, obwohl das SI-Präfix für eine Datenmenge von 1000 Byte steht. Am bekanntesten ist dieses Problem bei Festplatten: Hersteller geben die Speicherkapazität stets mit SI-Präfixen an, während die meisten Betriebssysteme den freien Speicher aber auf Basis von Binärpräfixen berechnen. Da Betriebssysteme schon beinahe grundsätzlich nicht die korrekten Abkürzungen für die Binärpräfixe verwenden, scheint es dem unwissenden Benutzer, als ob eine Festplatte mit einer Kapazität von 1 TB am Ende gerade einmal 931 GB hat – bei letzterer Angabe handelt es sich aber um ein Binärpräfix, korrekterweise also 931 GiB.

Derzeit ist Mac OS X 10.6 das einzige Betriebssystem, welches Si-Präfixe anstelle der Binärpräfixe verwendet. Linux und Windows verwenden nach wie vor Binärpräfixe – und geben fälschlicherweise SI-Präfixe an. Zwar ist es letzten Endes Geschmackssache, welchen Präfixtyp man bevorzugt. Um aber im Rahmen der Vorlesung Missverständnisse zu vermeiden, unterscheiden wir strikt zwischen beiden Typen. Tabelle A.1 fasst die gebräuchlichen Abkürzungen und Bezeichnungen beider Präfixtypen zusammen.

SI Präfix	**Bezeichnung**	**Faktor**	**Binärpräfix**	**Bezeichnung**	**Faktor**
k	kilo	10^3	Ki	Kibi	2^{10}
M	Mega	10^6	Mi	Mebi	2^{20}
G	Giga	10^9	Gi	Gibi	2^{30}
T	Tera	10^{12}	Ti	Tebi	2^{40}
P	Peta	10^{15}	Pi	Pebi	2^{50}
E	Exa	10^{18}	Ei	Exbi	2^{60}
Z	Zeta	10^{21}	Zi	Zebi	2^{70}

Tabelle A.1.: Übersicht über SI- und Binärpräfixe.

A.2. Fourierreihen: Ein Beispiel

Gesucht sei die Reihenentwicklung der in Abbildung A.1 dargestellten Rechteckfunktion

$$f(t) = \begin{cases} 1 & (n-1)\cdot T \le t < \left(n-\frac{1}{2}\right)\cdot T \\ -1 & \left(n-\frac{1}{2}\right)\cdot T \le t < T \end{cases}, \quad \forall n \in \mathbb{Z}.$$

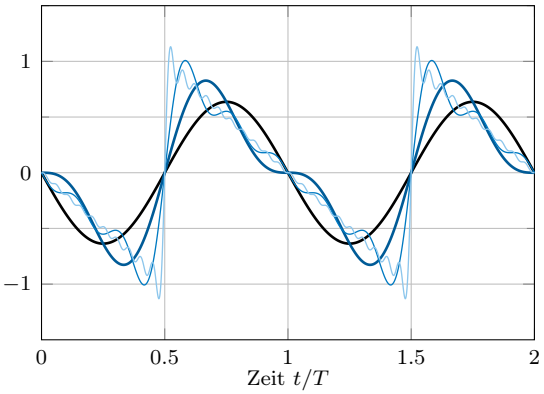


Abbildung A.1.: Periodischer Rechteckimpuls

Ein Blick auf die Funktion $f(t)$ lässt erwarten, dass die Koeffizienten a_k null sind, da diese die Gewichte für Kosinus-Schwingungen darstellen. Der Kosinus ist jedoch an ganzzahligen Vielfachen von π (minus) eins und an den Stellen $\frac{1}{4}n\pi$ und $\frac{3}{4}n\pi$ gleich null. Da die Funktion $f(t)$ hingegen an den Stellen $\frac{1}{4}n\pi$ und $\frac{3}{4}n\pi$ gleich eins bzw. minus eins ist, können Kosinus-Schwingungen keinen Beitrag liefern. Dies lässt sich mittels Formel 1.1 überprüfen:

$$
\begin{aligned}
a_k &= \frac{2}{T} \cdot \int_0^T s(t)\cos(k\omega t)\ dt \overset{T\equiv 1}{=} 2 \cdot \int_0^{1/2} \cos(k\omega t)\ dt - 2 \cdot \int_{1/2}^{1} \cos(k\omega t)\ dt \\
&= \frac{2}{k\omega} \cdot \left[\sin(k\omega t)\right]_0^{1/2} - \frac{2}{k\omega} \cdot \left[\sin(k\omega t)\right]_{1/2}^{1} \\
&= \frac{2}{k\omega} \cdot \left[\sin\left(\frac{k\omega}{2}\right) - 0\right] - \frac{2}{k\omega} \cdot \left[\underbrace{\sin(k\omega)}_{=0\ \forall k\in\mathbb{Z}} - \sin\left(\frac{k\omega}{2}\right)\right] \\
&\overset{\omega=2\pi/T}{=} \frac{2}{k\pi} \underbrace{\sin\left(k\pi\right)}_{=0\ \forall k\in\mathbb{Z}} = 0.
\end{aligned}
$$

Für die Koeffizienten b_k gilt erhalten wir:

$$
\begin{aligned}
b_k &= \frac{2}{T} \cdot \int_0^T s(t)\sin(k\omega t)\ dt \overset{T\equiv 1}{=} 2 \cdot \int_0^{1/2} \sin(k\omega t)\ dt - 2 \cdot \int_{1/2}^{1} \sin(k\omega t)\ dt \\
&= -\frac{2}{k\omega} \cdot \left[\cos(k\omega t)\right]_0^{1/2} + \frac{2}{k\omega} \cdot \left[\cos(k\omega t)\right]_{1/2}^{1} \\
&= -\frac{2}{k\omega} \cdot \left[\cos\left(\frac{k\omega}{2}\right) - 1\right] + \frac{2}{k\omega} \cdot \left[\underbrace{\cos(k\omega)}_{=\cos(2\pi k)=1\ \forall k\in\mathbb{Z}} - \cos\left(\frac{k\omega}{2}\right)\right] \\
&\overset{\omega=2\pi/T}{=} \frac{2}{k\pi} \cdot (1 - \cos(k\pi)) \overset{\frac{1}{2}(1-\cos(2x))=\sin^2(x)}{=} \frac{4}{k\pi}\sin^2\left(\frac{k\pi}{2}\right)
\end{aligned}
$$

Mit

$$\sin^2\left(\frac{k\pi}{2}\right) = \begin{cases} 0 & k = 0, 4, 8, 12, \ldots \\ 1 & k = 1, 5, 9, 13, \ldots \\ 0 & k = 2, 6, 10, 14 \\ 1 & k = 3, 7, 11, 15, \ldots \end{cases}$$

folgt:

$$b_k = \begin{cases} \frac{4}{\pi} \cdot \frac{1}{k} & \text{für } k = 1, 3, 5, \ldots \\ 0 & \text{sonst} \end{cases}.$$

Damit ist die Reihenentwicklung gegeben als

$$s(t) = \frac{a_0}{2} + \sum_{k=1}^{\infty} \left(a_k \cos(k\omega t) + b_k \sin(k\omega t)\right) \tag{A.1}$$

$$= \frac{4}{\pi} \cdot \left(\sin(\omega t) + \frac{1}{3}\sin(3\omega t) + \frac{1}{5}\sin(5\omega t) + \ldots\right) = \frac{4}{\pi} \cdot \sum_{n=1}^{\infty} \frac{\sin((2n-1)\omega t)}{2n-1}. \tag{A.2}$$

Literaturverzeichnis

[1] CAIDA, IPv4 Census Map, http://www.caida.org/research/id-consumption/#ipv4-census-map.

[2] DE-CIX, IP Traffic Statistics, http://www.de-cix.net/about/statistics.

[3] Jeffrey Dean, Sanjay Ghemawat, and Google Inc, Mapreduce: simplified data processing on large clusters, In OSDI'04: Proceedings of the 6th conference on Symposium on Opearting Systems Design and Implementation, USENIX Association, 2004.

[4] Christian Grothoff, Krista Grothoff, Matthew J. Rutherford, Kai Christian Bader, Harald Meier, Craig Ritzdorf, Tilo Eissler, Nathan Evans, and Chris GauthierDickey, Dup: A distributed stream processing language, IFIP International Conference on Network and Parallel Computing (Zhengzhou, China), Springer Verlag, Springer Verlag, 2010.

[5] Kai Li and Paul Hudak, Memory coherence in shared virtual memory systems, 1989.

[6] L. L. Peterson and S. Davie B., Computer Networks – A System Approach, 4. ed., ch. Internetworking, pp. 234 – 242, Elsevier, 2007, Auszug s. Moodle/SVN.

[7] _____, Computer Networks – A System Approach, 4. ed., ch. Internetworking, pp. 248 – 256, Elsevier, 2007, Auszug s. Moodle/SVN.

[8] _____, Computer Networks – A System Approach, 4. ed., ch. Internetworking, pp. 259 – 262, Elsevier, 2007, Auszug s. Moodle/SVN.

[9] E. Stein, Taschenbuch Rechnernetze und Internet, 2. ed., ch. Codierung und Modulation, pp. 59–66, Fachbuchverlag Leipzig, 2004, Auszug s. Moodle/SVN.

[10] _____, Taschenbuch Rechnernetze und Internet, 2. ed., ch. Konzepte: Lokale Netzwerke, pp. 191–218, Fachbuchverlag Leipzig, 2004, Auszug s. Moodle/SVN.

[11] _____, Taschenbuch Rechnernetze und Internet, 2. ed., ch. Fehlererkennung durch CRC, pp. 86–87, Fachbuchverlag Leipzig, 2004, Auszug s. Moodle/SVN.

[12] _____, Taschenbuch Rechnernetze und Internet, 2. ed., ch. Netzaufbau, pp. 200–203, Fachbuchverlag Leipzig, 2004, Auszug s. Moodle/SVN.

[13] Andrew S. Tanenbaum and Maarten van Steen, Verteilte systeme, 2., aufl. ed., PEARSON STUDIUM, 2007.

[14] M. Werner, Nachrichtentechnik – eine Einführung für alle Studiengänge, 6. ed., ch. Digitale Signalverarbeitung und Audio-Codierung, pp. 72 – 80, Vieweg + Teubner, 2007, Auszug s. Moodle/SVN.

[15] _____, Nachrichtentechnik – eine Einführung für alle Studiengänge, 6. ed., ch. Digitale Übertragung im Basisband, pp. 127 – 136, Vieweg + Teubner, 2007, Auszug s. Moodle/SVN.

[16] _____, Nachrichtentechnik – eine Einführung für alle Studiengänge, 6. ed., Vieweg + Teubner, 2007.