Einführung

Was ist das Internet?

- **Komponentensicht**:
  1. *Hosts*: führen Netzwerkanwendungen aus
  2. *Kommunikationsmedien*: Kupferkabel, Glasfaser, Funk
  3. *Zwischensysteme*: Weiterleitung durch Router und Switches
- **Dienstsicht**: Infrastruktur, die Dienste für *verteilte* Anwendungen bereitstellt
  - *Kommunikation* (Mail, Messaging, soziale Medien)
  - *Information* (Surfen)
  - *Unterhaltung* (Streaming, Spiele)
- **Protokolle**: definieren Regel und Formate für Kommunikation

Rand des Internet

- **Endsysteme**: Clients, Server
- **Zugangsnetze**: Heimnetz, Mobiles Zugangsnetz, Unternehmensnetz

Kern des Internet

- **Pakete**: Voneinander unabhängige Einheiten für die Weiterleitung, werden durch das Netz zur Zielanwendung geleitet
- **Interne Struktur**:
  - Zugangs-ISPs verbunden mit Globalen Tier 1 ISPs
  - Verknüpft durch peering links und an IXPs (Internet Exchange Point)
  - Dazu Content Provider Networks und Regionale Netze

Internet-Historie

- **Paradigmenwechsel**: Telefonnetz (Leitungsvermittelt, Zentral)
  $\Rightarrow$ Internet (Paketvermittelt, Dezentral)
- **Anfang**: ARPAnet (1969), dann weitere Netze, Protokolle (zunächst Universitäten, dann zunehmende Kommerzialisierung)
- **Entwurfsprinzipien**: Minimalism/Autonomy, Best Effort Service, Stateless Routers, Decentralized Control

Anwendungsschicht

Historie

- **70er/80er**: Textbasierte Anwendungen (EMail, Remote Access)
- **90er**: World Wide Web, Instant Messaging, P2P-Filesharing
- **seit 2000**: steigende Vielfalt + Allgegenwärtigkeit: ($\Rightarrow$ Kritische Infrastruktur) Voice over IP, Streaming, Gaming, Soziale Netzwerke, Smartphones

Schichtenmodell

- **Prozess**: Programm, das im Endsystem (Anwendungsschicht) abläuft
- **Nachricht**: Ausgetauscht zwischen Prozessen auf *unterschiedlichen* Endsystemen
- Kommunikation in Schichten organisiert
- **Anwendungsschicht**: oberste Schicht
  - enthält Anwendungsprotokolle
  - Anwendung kümmert sich nicht um Datentransport
- **Datentransport**: unter Anwendungsschicht liegende Schichten
  - Interna für Anwendung transparent
  - Aber: Anwendung merkt Verzögerungen (Latenzen)

Verzögerung (Latenz)

- **Ausbreitungsverzögerung** $t_a = \frac{d}{v}$
  - Zeitspanne zwischen Absenden eines Signals und dessen Eintreffen am anderen Ende des Mediums
  - Abhängig von: Ausbreitungsgeschwindigkeit v, Länge des Mediums d
- **Sendezeit** $t_s = \frac{X}{r}$
  - Zeit zwischen Beginn und Abschluss der Sendung
  - Abhängig von: Datenmenge X, Datenrate (Durchsatz) des Mediums r
  - **Achtung**: Nach Sendungsabschluss sind die Daten noch nicht beim Empfänger! $\rightsquigarrow$ Ausbreitungsverzögerung t_a
- **Verzögerung im Router**:
  - *Pufferung* der Daten in Warteschlange
  - *Verarbeitung* (Fehlerüberprüfung usw.)

Protokollstack

- **Application**: SMTP, HTTP, XMPP, …
- **Transport**: TCP, UDP
- **Network**: IP
- **Data Link**: Ethernet, 802.11 (WiFi)
- **Physical**: Bits auf Medium

Socket und Interface

- *Programmierschnittstelle für verteilte Anwendungen*
- Von OS bereitgestellte API
- Anwendungsprozess sendet/empfängt Nachrichten zum/vom Socket
- **Portnummern**: (De-) Multiplexing auf Endsystemen, viele Prozesse auf einem Endsystem kommunizieren gleichzeitig über Netzwerk
  $\rightsquigarrow$ eindeutige Socket-Identifikation über Portnummer

Client-Server-Anwendungen

- **Server**: Ständig in Betrieb, permanente IP-Adresse, häufig in Datenzentren
- **Clients**: Kommunizieren mit Server, *nicht* direkt miteinander, dyn. IP-Adresse

Peer-to-Peer-Anwendungen

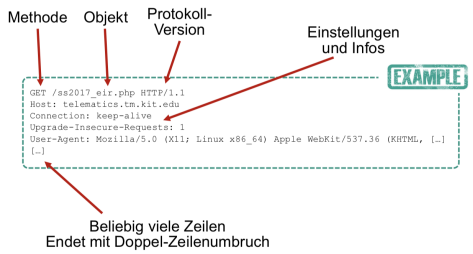
- Endsysteme kommunizieren direkt miteinander
  - Fordern Dienste von anderen Peers an und stellen selbst Dienste bereit
  - Nicht permanent verbunden, wechseln dynamisch IP-Adressen
    $\rightsquigarrow$ komplexes Management
- **Selbstskalierend**: Neue Peers erhöhen Kapa, fordern aber auch Dienste an

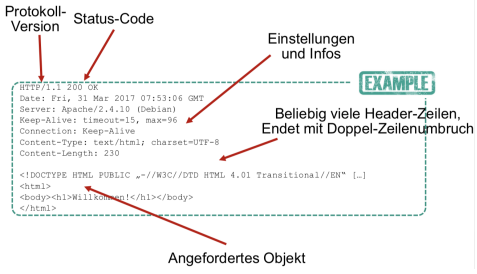
Web und HTTP — Web-Dokumente

- Webseiten bestehen aus HTML-Datei und anderen Objekten (.js, .png,…)
- Jedes Objekt über URL (*uniform resource locator*) referenzierbar

HTTP (Hypertext Transfer Protocol)

- ASCII-basiertes Transferprotokoll der Anwendungsschicht im Web
- Basiert auf Client/Server-Modell
  - *Client (Request)*: Browser, der Web-Objekte anfordert
  - *Server (Response)*: Sendet über HTTP angeforderte Objekte
- Zustandslos: Jeder Request individuell, keine Zustandsinformation auf Server
- Kommunikation per TCP:
  1. Client initiiert Verbindungsaufbau (Standard-Port: 80)
  2. Server akzeptiert Verbindung
  3. Austausch von HTTP-Nachrichten
  4. Abbau der TCP-Verbindung
- HTTP-Anfragen können verschiedene **Methoden** nutzen:
  - **GET**: Ressource von Server zu Client übertragen (z.B. normale Webseite)
  - **POST**: Daten zu Ressource übertragen (z.B. Web-Formular)
  - PUT: neue Ressource anlegen
  - DELETE: Ressource löschen
  - HEAD: wie GET, aber nur HTTP-Header übertragen
- **Status-Codes:** Verarbeitungsindikator (Erfolg/Fehlschlag + Gründe)
  - **200**: Erfolg; Antwort ist in dieser Nachricht
  - **301**: Angefragtes Objekt verschoben (neue URL in Nachricht spezifiziert)
  - **400**: Server hat Anfrage nicht verstanden
  - **404**: Angefordertes Objekt existiert nicht
  - **505**: HTTP-Version nicht unterstützt





HTTP — Verbindungen

- **Round Trip Time (RTT)**: Zeit, die Paket von Sender zu Empfänger und zurück benötigt
- **Non-persistent HTTP**: Höchstens ein Objekt über eine TCP-Verbindung, danach geschlossen ⇝ Mehrere Objekte erfordern mehrere TCP-Verbindungen (evtl. parallel)
  - *Antwortzeit*: $2 * \text{RTT} + t_s$ pro Objekt (1 RTT Verbindungsaufbau, 1 RTT HTTP-Anfrage und erste Antwortbytes, t_s für Senden des Objekts)
- **Persistent HTTP**: Mehrere Objekte über eine TCP-Verbindung
  - *Antwortzeit*: Nur eine RTT für nachfolgende Objekte

HTTP — Cookies

- *Speichert Nutzer-Server-Zustand* ⇝ Server kann Inhalt pro Nutzer bereitstellen
- **Komponenten**:
  - Cookie-Information in HTTP-Response-Nachricht (*Set-Cookie*)
  - Cookie-Information wird in nachfolgenden HTTP-Requests genutzt
  - Datei mit Cookies wird auf Nutzer-Endsystem vom Browser verwaltet
  - Datenbank bei Webseite: Server muss Cookies richtig interpretieren können
- **Privatsphäre:** Webseiten unterscheiden Nutzer durch Cookies
  ⇝ Werbeanbieter können Nutzer über viele Seiten tracken

Mail — Komponenten

- **User Agent** (UA): Lesen, senden, weiterleiten
- **Mailserver**: User-Mailboxen, *mail transfer/mail delivery agent* (MTA/MDA)

Mail — SMTP (Simple Mail Transfer Protocol)

- Transfer von Mails zwischen Mailservern/von User Agent zu Mailserver
- **Phasen**: Handshake, Nachrichtenübermittlung (Header + Body), Abschluss
- Client/Server-Model, Command/Response-Interaktionen
  - Ähnlich Request/Response bei HTTP, nutzt ebenfalls TCP (Port 25)
  - Kommandos: ASCII-Text
  - Antwort: Statuscode + Nachricht

Mail — MIME

= *Multipurpose Internet Mail Extensions*

- **Problem**: SMTP kann nur 7-Bit ASCII-Texte versenden, keine Dateien
- **MIME**: erweitert Kopfteil einer Nachricht um Formatinformation
- **Content-Type**: Definiert Typ des E-Mail-Inhalts; *Content-Transfer-Encoding*

Mail — Postfach-Abfrage

- **POP3** (*post office protocol 3*): Verwaltung im UA, *keine* Synchronisation
  - Client holt am Mailserver gespeicherte Nachrichten ab
  - nur einfache Funktionalität (*list, retr, dele*)
- **IMAP** (*interactive mail access protocol*): Zentrale Verwaltung auf Mailserver, erweiterte Kommandos (Ordner, Filter)
- **Web-Mail**

WhatsApp — XMPP

= *eXtensible Messaging and Presence Protocol*, Echtzeit XML-Streaming

- Dezentral, ähnlich wie E-Mail
- Whatsapp nutzt Zentralen Server und proprietäre Variante des Protokolls
- **Clients**: zu ihrem jeweiligen Server verbunden
- **Server**: verbinden sich untereinander zur Nachrichtenübermittlung
- **Nachricht:** XML-Dokumente (erweiterbares Format)
- **Adressformat**: Server + Username, evtl. Client (z.B. alice@jabber.org/laptop)

DNS (Domain Name System)

- **Ziel**: Verwendung von Namen statt IP-Adressen
- **Aufgabe**: Zuordnung IP-Adresse ↔ Name (Subdomäne.Domäne.TLD)
- **Funktionalitäten**:
  - *Registrierung* von Namen + IP-Adressen
  - *Auflösung* von Namen in IP-Adressen
  - *Host Alias*: Löse einfachere Alias-Namen in kanonische Namen auf
  - *Mail Server Alias*: Liefere E-Mail-Server zu einer Domain
  - *Lastverteilung*: Mehrere IPs redundanter Server in zufälliger Reihenfolge
- Protokoll der Anwendungsschicht, über UDP realisiert, Client-Server-Modell
- Basisdienst, keine eigentliche Anwendung: Komplexität am Rand des Netzes

DNS — Aufbau

- Probleme mit zentralem Server: Singuläre Fehlerquelle, Verkehrsaufkommen, geografische Entfernung, Verwaltungsaufwand, Abhängigkeit
- Verteilte Datenbank in einer **Hierarchie** von Name-Servern (DNS-Servern)
  - *Lokaler NS*: Erste Anfrage immer zu lokalem Server, Antwort aus eigener Zuordnungsdatenbank, Cache oder nach Befragung anderer DNS-Server
  - *Autoritativer NS*: Enthält autoritative Abbildungen, jeder Host ist bei einem registriert (in seinem Netz)
  - *Top-Level Domain (TLD) Server*
  - *Root-Server*: Enthalten nur TLD-Einträge, fixe IPs, 13 Root-Server-Cluster

DNS — Anfragen

- **Rekursiv**: kennt angefragter Server Antwort nicht, fragt dieser weitere Server, bis er Antwort zurückliefern kann
- **Iterativ**: kennt Server die Antwort nicht, verweist er *Client* an andere Server
- **Üblich**: Client fragt lokalen Name-Server rekursiv, dieser dann iterativ

DNS — Resource Records (RR)

- DNS ordnet Domänen zu Einträgen zu
- **A / AAAA** (Address): Abbildung Name auf IPv4/IPv6-Adresse
- **MX** (Mail Exchange): Mailserver einer Domäne (IP-Adresse)
- **NS** (Name Server): Nameserver einer Domäne (Hostname)
- **CNAME** (Canonical Name): Alias-Namen für Rechner/Domänen (Domain)
- **PTR** (Pointer): Abbildung IP-Adresse auf Name (Domain)

Content Delivery Networks (CDN)

- **Beispiel: Videostreaming** (hohe Datenrate, Datenqualität)
- **DASH (Dynamic, Adaptive Streaming over HTTP)**: Video aufgeteilt in Chunks, jeweils in mehreren Qualitäten (Bitraten) verfügbar, URLs und Infos in Manifest-Datei
  Client wählt adaptiv bestmögliche Bitrate für jeden Chunk
- **Content Distribution**: Content zu hunderttausend Nutzern bringen
- Mega-Server: Skaliert nicht (Single Point of Failure, Netzwerküberlastung, Entfernung)
- **CDN:** Content auf geographisch verteilte Server kopieren
  Third-Party CDNs (z.B. Akamei, Limelight), Private CDNs (z.B. Google für YouTube)
- **Strategien**:
  - *enter deep*: Viele kleine Cluster in Zugangsnetzen nahe beim Nutzer
  - *bring home*: Wenige große Cluster in wichtigen IXPs für geringeren Verteilungs- und Wartungsaufwand
- DNS-Manipulation: Autoritativer DNS-Server des CDN passt Antwort an IP-Adresse des anfragenden lokalen DNS-Server an, wählt einen nahe gelegenen CDN-Server aus

Transportschicht

Ziele und Prinzipien

- Kommunikation zwischen Prozessen
- *Verbergen von Transportdetails vor höheren Schichten*
- *Bereitstellung von Transportdiensten*
  ⇝ logische **Nutzer-zu-Nutzer-Kommunikation** (Anwendungen)
  - Vermittlungsschicht: Ende-zu-Ende-Kommunikation (Endsysteme)
- Transportprotokoll läuft auf Endsystemen
- **Sender**:
  - *Segmentieren* von Anwendungsnachrichten
  - *Weiterleiten* an Vermittlungsschicht
- **Empfänger**:
  - *Reassemblieren* der Segmente in Nachrichten
  - *Weiterleiten* an Anwendungsschicht

Protokolle

- **UDP** (*user datagram protocol*): verbindungsloser, *unzuverlässiger* Transport
- **TCP** (*transmission control protocol*): verb.-orientierter, *zuverlässiger* Transport
- **Unzuverlässig**: keine Fehlermaßnahmen; unklar, ob Daten korr. ankommen
- **Zuverlässig**: Fehlermaßnahmen garantieren
  - *Korrektheit*, *Vollständigkeit*, *Reihenfolge*
  - keine *Duplikate*, keine *Phantom-Pakete*

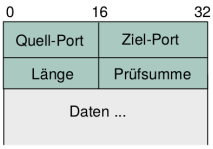
(De-)Multiplexing — Ports

- Ausliefern von Segmenten an den korrekten Socket (nutze Transportheader)

- Port = *Adresse der Transportschicht* (Kennzeichnung der Prozesse)
- Unstrukturierte Nummer (16 Bit), 0 bis 65535
- **Well known ports**: viele Portnummern unter 1024 für häufig benutzte Anwendungen reserviert (Telnet, HTTP, SMTP, FTP, …)
- Eindeutige Adressierung eines Prozesses: "IP-Adresse:Port"

UDP (User Datagram Protocol)

- *Sehr einfaches Transportprotokoll mit sehr geringem Overhead (8 Byte)*
- **Eigenschaften**:
  - (De-) Multiplexen von Segmenten für Prozesse
  - *Prüfsumme* für Bitfehler
  - *best effort*: keine Zusagen über Auslieferung bei Empfänger
  - *verbindungslos*
  - *keine Verbindungsaufbauphase*: sofortiges Senden ⇝ keine Verzögerung
  - *kein Verbindungszustand*: keine Verbindungsinformationen im Endsystem ⇝ skaliert z.B. für Server besser
  - *Unreguliertes Senden*: kann Daten so schnell senden wie von Anwendung geliefert und von Netz abgenommen
- **Verwendung**: **DNS**, Multimedia (VoIP)



Bitfehler

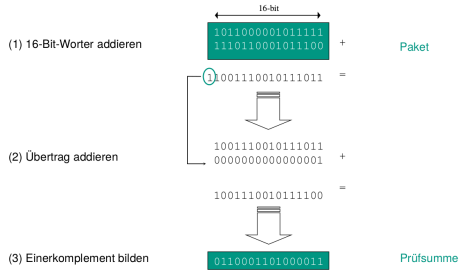
- *Verfälschung von Bits während dem Datentransport*
- **Ursachen**: Dämpfung, Übersprechen, Synchronisationsverlust, …
- **Einzelbitfehler**: ein *einzelnes* Bit fehlerhaft
- **Bündelfehler**: mehrere direkt aufeinanderfolgende Bits fehlerhaft
- **Bitfehlerrate**: Maß für Fehlerhäufigkeit = $\frac{\text{Summe gestörter Bits}}{\text{Summe übertragener Bits}}$

Bitfehler — Erkennung/Korrektur

- **Fehlererkennung** (*error detecting code*, EDC)
- **Fehlerkorrektur** (*forward error correction*, FEC)
- Hinzufügen von Redundanz (Paritätsbit ($d_{\min} = 2$), Prüfsumme, …)
  ⇝ Ausreichend stark unterscheidende Codewörter
- **Hamming-Abstand** $d_{i,j}$: #Stellen, an denen sich Codewörter unterscheiden
- Hamming-Abstand des Codes $d_{\min}$: Min Abstand zw Codewörtern
- Erkennen von bis zu $d_{\min} - 1$ Bitfehlern
- Korrigieren von bis zu $\lfloor \frac{d_{\min}-1}{2} \rfloor$ Bitfehlern
- Kontrollmatrix H, für übertragenes Wort w gilt:
  - Syndrom $S = w \cdot H^T = 0$, falls Übertragung fehlerfrei (w ist Codewort)
  - Ansonsten kann das Syndrom die Position des Bitfehlers angeben

Bitfehler — Internet-Prüfsumme (UDP, TCP, IP)

- **Prinzip**: Aufaddieren aller übertragenen Wörter (16 Bit, als Integer interpretiert), evtl. Übertrag addieren, bitweise negieren (Einerkomplement)
- **Nachteil**: Falsche Reihenfolge kann nicht erkannt werden



Paketfehler

- *Verlust / Duplizierung von Paketen*
- *Phantom-Pakete*
- *Reihenfolgenvertauschung*
- **Ursachen**: Zwischensystemüberlastung, Unterschiedliche Wege durch Netz, Verfrühte Datenwiederholung, …

Paketfehler — Erkennung/Korrektur

- **Sequenznummern** (*sequence number*):
  - Pakete werden durchnummeriert
  - Empfänger kann Vollständigkeit, Reihenfolge, Duplikate feststellen
- **Quittungen** (*acknowledgements*): Empfänger informiert Sender, ob Paket angekommen ist
  - *positiv*: Daten erhalten (ACK)
  - *negativ*: Daten *nicht* erhalten (NACK)
  - *selektiv*: Einzelnes Paket
  - *kumulativ*: Paketmenge (bis zu einer Sequenznummer)
- **Zeitgeber** (*timer*):
  - Sender merkt nicht, wenn ein Paket nicht angekommen ist
  - Nach zeitl Obergrenze wird *vermutet*, dass Paket nicht angekommen ist ⇝ *Sendewiederholung* (*retransmissions*)
- **Automatic Repeat Request (ARQ)**: Sendewiederholungsvariante: Sender erhält positive Quittungen, kann Sendewiederholungen ausführen

ARQ — Stop-and-Wait

- **Prinzip**: Sender wartet auf Quittung nach jedem gesendeten Paket
  - Erst *nach* Quittungserhalt wird nächstes Paket gesendet
  - Keine Quittung nach Wartezeit (Zeitgeber) ⇝ Sendewiederholung
- **Sequenznummern**: 1 Bit (Empfänger kann nur Paket doppelt erhalten)
- Sehr einfaches ARQ-Verfahren (z.B. im WLAN)
- **Auslastung**: $U = \frac{1}{1+2a}$ (mit $a = \frac{t_a}{t_s}$), $t_{Ges} = t_s + 2t_a$
- **a** = $\frac{t_a}{t_s}$ ist Verhältnis der Länge des Mediums in Bit zur Länge der Dateneinheit
- Bandbreitenverzögerungsprodukt $\frac{m}{v}r$: Länge des Mediums in Bit (Länge m, Ausbreitungsgeschwindigkeit v, Datenrate r)

ARQ — Go-Back-N

- **Zeil**: Leistungsfähigkeit im Vergleich zu Stop-and-Wait erhöhen
- **Prinzip**: Sender sendet mehrere Pakete bis Quittungspflicht
  - Begrenzte Anzahl (**Fenster**, *window*) an nicht quittierten Paketen
  - Quittierung durch kumulative Quittungen
- **Fehlerfall**:
  - Empfänger empfängt fehlerhaftes Paket, verwirft alle nachfolgenden Pakete
  - Sender wiederholt bei Ablauf des Zeitgebers alle nicht quittierten Pakete
- **Auslastung**: $U = \begin{cases} 1 & W \geq 1 + 2a \\ \frac{W}{1+2a} & \text{sonst} \end{cases}$

ARQ — Selective Repeat

- **Ziel**: Datenaufkommen von Go-Back-N reduzieren
- **Prinzip**: Wie Go-Back-N, Empfänger quittiert selektiv
- **Fehlerfall**: Empfänger puffert und bestätigt nachfolgende, korrekte Pakete ⇝ Sender wiederholt nur nicht korrekt empfangene Pakete
- **Selective Repeat**: Fehler-Paket bleibt unbestätigt, Sender wartet auf Timeout
- **Selective Reject**: Empfänger sendet für Fehler-Paket negative Quittung ⇝ Sender wiederholt sofort und wartet nicht auf Timeout

Paketfehler — Vorwärtsfehlerkorrektur

- **Ziel**: Empfänger muss nur drei von vier Paketen korrekt empfangen um fehlendes Paket rekonstruieren zu können
- **Prinzip**: XOR-Verknüpfung der drei Pakete ⇝ fehlendes Paket

Flusskontrolle

- **Problem**: *Überlastung* von Empfänger durch Sender ⇝ Datenverlust
- Sender muss Größe des Empfangspuffers berücksichtigen
- **Anforderungen**: einfach, fair, stabil, möglichst wenig Netzressourcen nutzen

Flusskontrolle — Halt-und-Weiter

- **Prinzip**: Empfänger sendet `halt` und `weiter`-Signal
- **Bewertung**:
  - nur auf Vollduplex verwendbar
  - nicht effektiv bei hohen Verzögerungen
  - Probleme bei Verlust der `halt`-Meldung
- **Beispiel**: Fast-/Gigabit-Ethernet

Flusskontrolle — Stop-and-Wait

- **Prinzip**: Empfänger sendet Quittung erst, wenn er wieder empfangen kann
- Sender wird durch Zurückhalten gebremst

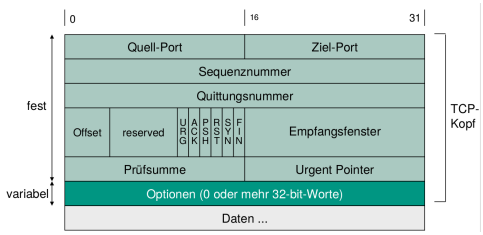
- **Problem**: Sender kann nicht zwischen Datenverlust und Überlastung unterscheiden

Flusskontrolle — Kreditbasiert

- **Prinzip**:
  - Sender kann höchstens n Pakete unquittiert senden
  - n = Pufferkapazität des Senders ⇒ **Sendekredit**
  - Alternativbezeichnung: Fenster (*sliding window*)
  - Fenster wird durch korrekte positive Quittung weitergeschaltet
  - Empfänger kann Kredit bestimmen (z.B. in TCP)

TCP — Prinzip

- **Zuverlässiger**, verbindungsorientierter Transportdienst zwischen Anwendungen
- *Erhält Bytestrom von Anwendung (über Socket), übergibt TCP-Segmente an IP*
- Wann wird ein TCP-Segment erzeugt und versendet?
  - MSS (*maximum segment size*): Anwendungsdatenlänge (z.B. 1460 Byte)
  - Push (`PSH` in TCP-Segmentkopf): Sender verlangt sofortiges Senden
  - Zeitgeber: Nach Zeitintervall der Inaktivität vorhandene Daten senden
- **Fehlerkontrolle**: Sequenznr., Prüfsumme, Quittierungen, Sendewdh-en
- **Sequenznummern**: pro Byte, nicht pro Segment (erstes Byte in Segment, initiale Sequenznummer von Endsystem *zufällig* gewählt)
- **Quittungen:** positive kumulative Quittungen (Sequenznummer des *nächsten* Bytes, das erwartet wird), mit Datensegment versendet (*piggybacked* im Header)

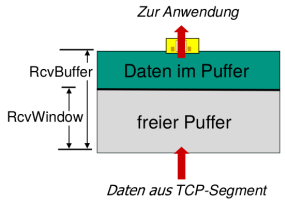


TCP — Felder

- **Quell-/Ziel-Port**: Identifizieren Verbindungsendpunkte
- **Sequenznummer**: gemessen in Byte, nicht pro Segment
- **Quittung**: nächste von Empfänger erwartete Sequenznummer
- **Offset**: Anzahl 32 Bit-Wörter in TCP-Kopf
- **URG**: 1, falls *urgent pointer* verwendet wird (idR leer)
- **SYN**: Bei Verbindungsaufbau *connection request* oder *connection confirmation*
- **ACK**: Gültigkeit des Quittungsfeldes; unterscheidet bei gesetztem SYN-Bit zwischen Request und Confirmation;
- **FIN**: Sender möchte nichts mehr senden
- **RST**: Verbindung zurücksetzen
- **PSH**: übergebene Daten sollen sofort weitergeleitet werden (idR leer)
- **Empfangsfenster**: Flusskontrolle
- **Prüfsumme**: Prüfsumme über TCP-Kopf, Daten und Pseudoheader
- **Urgent-Zeiger**: relativer Zeiger auf wichtige Daten
- **Optionen-Feld**: Optionen variabler Länge ($n * 32$ Bit)

TCP — Flusskontrolle

- **Ziel**: Empfänger nicht überlasten
- **Prinzip**: Empfänger reserviert Puffer pro Verbindung (expl Kreditvergabe)
  - `RcvBuffer`: gesamter Pufferplatz (default 4096 Byte)
  - `RcvWindow`: freier Pufferplatz (Empfangsfenster) in TCP-Header mitsenden
  - Sender sendet nicht mehr unbestätigt als in `RcvWindow` passt

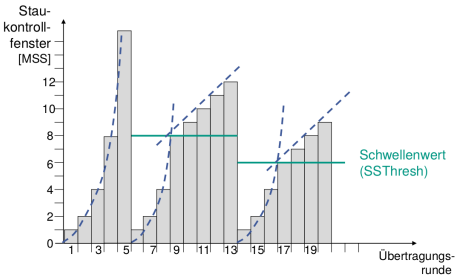


TCP — Verbindungen

- **Verbindungslos (UDP)**: Daten werden ohne vorherigen Handshake gesendet
  - *Vorteil*: schnelle Datenversendung möglich
  - *Nachteil*: kein Feedback, keine Bestätigung
- **Verbindungsorientiert (TCP)**: Expliziter Verbindungsaufbau/-abbau
  - *Vorteil*: Kommunikationsparameter können ausgehandelt werden
  - *Nachteile*: Verzögerter Datenaustausch, Overhead ggf größer als Daten
- Probleme mit 2-Wege-Handshake: Verzögerungen + Wiederholte Nachrichten können zu halboffenen Verbindungen führen
- **3Way-Handshake**: SYN, SYN/ACK, ACK (letztes ACK kann Nutzdaten enth)
- **Abbau**: Richtungen unabhängig jeweils durch FIN, ACK schließen, danach Warten vor vollständigem löschen (falls Wiederholungen auftreten)
- **Abbruch**: RST, Verbindung wird unmittelbar geschlossen

TCP — Staukontrolle

- **Ziel**: Netz nicht überlasten (Pufferüberläufe in Routern vermeiden)
- **Staukontrollfenster** (*congestion window*, `CWnd`) beschränkt maximale Datenmenge: LastByteSent − LastByteAcked ≤ min{CWnd, RcvWindow}
- **Stauerkennung**: Vermutung einer Stausituation bei ausbleibender Quittung
- **Staubehebung**: Reduktion von `CWnd`
- Langsames Erhöhen von `CWnd` ⇝ herantasten an Netzkapazität
- **Start**: `CWnd` = 1 `MSS` (*maximum segment size*)
- **Slow-Start**, falls `CWnd` ≤ `SSTresh` & Quittungen rechtzeitig da ⇝ *exponentielles* Erhöhen `CWnd` (`CWnd` += 1 bei jeder empfangenen Quittung)
- **Congestion Avoidance**, falls `CWnd` > `SSTresh` & Quittungen rechtzeitig da ⇝ *lineares* Erhöhen `CWnd` (`CWnd` += $\frac{1}{\texttt{CWnd}}$ bei jeder empfangenen Quittung)
- **Congestion**, falls Quittung nicht rechtzeitig da: Stau vermutet
  - `SSTresh` = max($\frac{\text{FlightSize}}{2}$, 2 * MSS) (FlightSize = unquittierte ges Daten)
  - `CWnd` zurücksetzen (neue Slow-Start-Phase): `CWnd` = 1 `MSS`



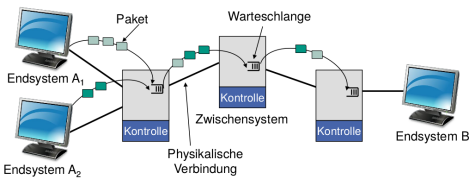
Vermittlung

Leitungsvermittlung

- Verbindung ist *durchgehender Kanal* mit konstanter Bandbreite für exklusive Nutzung
- **Multiplexing**: *starres Multiplexing* möglich
  - *Frequenzmultiplex*: Feste Zuweisung eines Frequenzabschnitts
  - *Zeitmultiplex*: Feste Zuweisung eines Zeitschlitz (*time slot*) in jedem Frame
- **Eigenschaften**:
  - Aufbau eines durchgehenden Übertragungskanals zwischen Endsystemen
  - Zwischensysteme: Zustandshaltung statt Adressinformation nötig
  - zugesicherte, feste Datenrate ⇝ ungenutzte Ress bei Nichtverwendung
  - Übertragungsverzögerungen nur physikalisch bedingt ⇝ keine Schwankungen durch Puffer
  - Reihenfolgentreue Bitfolgenübertragung
- **Einsatzgebiete**: Telefonnetze, GSM

Paketvermittlung

- **Prinzip**: Weiterleitung anhand von Kontrollinformation in Paket (Zieladresse in Datagrammen, lokale Kennung bei virtuellen Verbindungen)
- Zwischensysteme Speichern Pakete in *Warteschlangen* ⇝ Paketverlust möglich, variable Verzögerungszeiten
- Üblicherweise Zeitmultiplex (keine Reservierung von Ressourcen)
- **Varianten**:
  - *verbindungslos*: Datagramme
  - *verbindungsorientiert*: virtuelle Verbindungen



Paketvermittlung — Datagramme

- Pakete (= Datagramme) werden als isolierte Einheiten betrachtet
- **Zieladresse** in jedem Datagramm ⇝ keine Verbindungsverwaltung nötig
- **Routing**: Pakete können verschiedene Wege im Netz nehmen
  ⇝ Datagramme können bei Empfänger unsortiert ankommen

Paketvermittlung — virtuelle Verbindungen

- Fester Übertragungsweg zwischen zwei Endsystemen
- **Reihenfolgetreue**: Alle Pakete nehmen den selben Weg
- **Kennung**: *virtual circuit identifier* (VCI) für jeden Übertragungsabschnitt
- **Zieladresse** nur bei Aufbau nötig, Vermittlungsinfo in Zwischensystemen
- **Ablauf**: Verbindungsaufbau → Datenübertragung → Verbindungsabbau

Nachrichtenvermittlung

- *Vermittlung von Anwendungsnachrichten*
- Vermittlung üblicherweise mittels mehrerer Pakete
  ⇝ *Segmentierung* und *Reassemblierung* in Zwischensystemen (alle Teile müssen jeweils an gleiches System weitergeleitet werden)
  ⇝ *Ende-zu-Ende-Verzögerung wesentlich höher als bei Paketvermittlung*

Vermittlungsschicht — Überblick

- **Ende-zu-Ende**: transportiert Segmente zwischen Endsystemen
- Sender: Kapselt Segmente in Datagramme
- Empfänger: Segmente werden an Transportschicht ausgeliefert
- Protokolle in *allen* Endsystemen und Routern
- **Router** werten Felder im Kopf aller Datagramme aus, die sie passieren

Forwarding und Routing

- **Forwarding** (*Weiterleitung*): Datenebene
  - Pakete werden von Eingang an passenden Ausgang (Tabelle) weitergeleitet
  - Funktionen lokal in Router
- **Routing** (*Wegewahl*): Kontrollebene
  - Ermittelt Wege für Pakete und darauf basierend Weiterleitungstabelle
  - Betrachtet gesamtes Netz, benötigt Routingalgorithmus und -protokoll
- **Konzepte**:
  - traditionell: in jedem Router implementiert, interagieren auf Kontrollebene
  - *software defined networking* (SDN): in log zentralen Servern implementiert

Vermittlungsschicht — Begriffe

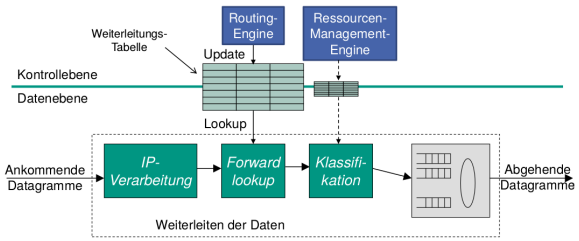
- **Router**: Auf Vermittlungsschicht operierendes Zwischensystem
  - leitet Datagramme mithilfe von Weiterleitungstabelle weiter
  - tauscht über Routingprotokolle Informationen mit anderen Routern aus
- **Route**: Weg eines Datagramms: Sequenz von Routern zwischen zwei Endsystemen
- **Link**: Übertragungsabschnitt zwischen 2 Routern (kann z.B. auch Brücken enthalten)
- **Port**: Eingangs-/Ausgangs-Netzwerkschnittstelle

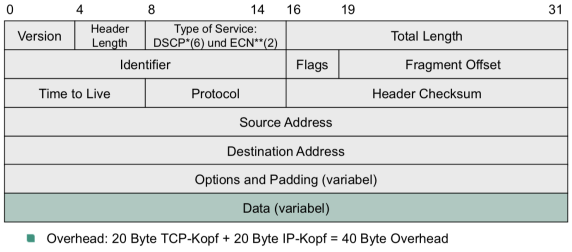
Vermittlungsschicht — Protokolle

- **IP** (*internet protocol*): unzuverlässige Datagrammübertragung
- **ICMP** (*internet control message protocol*): Kontrollinformationsaustausch innerhalb der Vermittlungsschicht
- **ARP** (*address resolution protocol*): Zuordnung von IP-Adressen zu Adressen der Sicherungsschicht
- **RARP** *reverse ARP*: Umkehrfunktionen von ARP
- **BGP** (*border gateway protocol*), **RIP** (*routing information protocol*), **OSPF** (*open shortest path first*): Routingprotokolle

Vermittlung — IP (Internet Protocol)

- Verbindungsloser, unzuverlässiger Vermittlungsdienst (kein Kontext in Zwischen- oder Endsystemen)
- IP macht die ganze Vermittlung ⇝ nur ein einziges Vermittlungsprotokoll
  - Interoperabilität erhöhen: Anzahl unterschiedlicher Interfaces verringern
  - Kleinster gemeinsamer Nenner: Anzahl nutzbarer Netze maximieren





Overhead: 20 Byte TCP-Kopf + 20 Byte IP-Kopf = 40 Byte Overhead

IP — Fragementieren + Reassemblieren

- Anpassung an Maximallänge unterl Netze (**MTU**: *maximum transfer unit*)
- Fragmentieren überall möglich, Reassemblieren nur in Endsystem
- **Fragment-Offset** (Einheit: 8 Bytes) + Flag-Felder im IP-Kopf

IP — Weiterleitung

- **Endsystem**:
  - Rechner mit Zieladresse direkt verbunden → Datagramm direkt zustellen
  - Sonst: Datagramm-Übergabe an Standardrouter
- **Weiterleitungstabelle**: Für jede Zieladresse (Endsystem- oder Netzadresse)
  - *Next-Hop-Router* (falls nicht im gleichen Netz)
  - *Netzschnittstelle*, an die Paket weitergeleitet wird (Schnittstelle, an der Next-Hop bzw. Endsystem hängt)
  - Flags

IP — Empfangsprozess

- **Überprüfungen**:
  - *Kopflänge*, *Datagrammlänge*, *Prüfsumme*
  - *Versionsnummer* IP, *Protokoll-Identifikation*
  - *Lebenszeit*, *Adressklassen*
- **Fehlerfall**: Benachrichtigung ICMP (*internet control message protocol*) — möglicherweise wird ICMP-Paket ausgesendet

IP — Adressierung

- **Ziel**: Eindeutige Identifizierung aller Interfaces von Routern/Endsystemen
- **IP-Adressen**: Adressen der Vermittlungsschicht – Kennung für Interfaces
  - *IPv4*: 32 Bit (z.B. 207.142.131.235)
  - *IPv6*: 128 Bit (z.B. 2001:0db8:85a3:08d3:1319:8a2e:0370:7344)
- IP-Adresse unterteilt in
  - *Subnetz-Teil*: high order bits
  - *Endsystem-Teil*: low order bits
- **Subnetz**: Interfaces mit selbem Subnetz können ohne Router kommunizieren
- **CIDR** (Classless Inter-Domain Routing): Subnetz-Teil kann unterschiedlich lang sein (Format: a.b.c.d/x, x = Anzahl Bits im Subnetz-Teil, z.B. 200.23.16.0/24)

Adresszuteilung

- **Provider**: Erhält Block von **ICANN** (*internet corporation for assigned names/numbers*), allokiert IPs, verwaltet DNS, weist Domainnamen zu
- Netz bekommt Subnetz-Teil von seinem Provider zugeordnet
- **Manuell**: Durch Systemadministrator
- **Dynamisch**: DHCP-Server liefert auf Anfrage IP-Adresse für Client

DHCP (dynamic host configuration protocol)

- Dynamischer Bezug von IP-Adressen durch Endsysteme
- Beschränkte zeitliche Gültigkeit (*lease*)
- Zusätzlich Subnetzmaske, Adresse Default-Gateway, DNS-Server, …möglich
- **Ablauf**: [Discover, Offer], Request, Ack (jeweils als Broadcast)

Internet Control Message Protocol (ICMP)

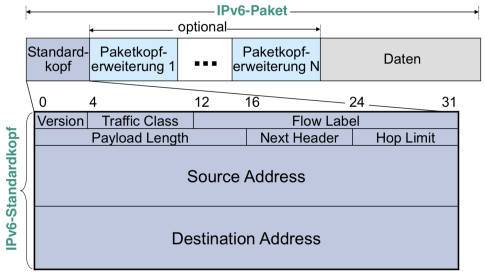
- Einzelne Datagrammverluste meldet IP nicht (unzuverlässiger Dienst)
- Schwerwiegende Probleme (z.B. Leitungsunterbrechung): Mitteilung via ICMP

⇒ ICMP tauscht Fehlernachrichten, Statusanfragen, Zustandsinfos aus

- **Echo** + Antwort (*echo and echo reply*):
  - Aktivitätsüberprüfung von Kommunikationssystemen
  - Empfänger Echo-Anfrage sendet erhaltene Daten in Echo-Antwort zurück
- **Zeitstempel** + Antwort (*timestamp and timestamp reply*): Bestimmung von Umlaufzeiten (*round trip time*, RTT)

IPv6

- **Problem**: Adressraum IPv4 geht aus ⇒ Erhöhung Adresslänge 32 auf 128 Bit
- Optimiere **Format des Headers** für schnelle Verarbeitung:
  - feste Kopflänge (40 Byte), Optionen als Erweiterungsköpfe (*next header*)
  - keine Unterstützung von Fragmentierung, keine Prüfsumme
- ICMPv6: neue Version von ICMP



Routing — Prinzipien

- **Ziel**: Guten Weg durch Netz finden (geringste Kosten)
- **Netzgraph**: Netz wird als Graph verstanden
  - *Knoten*: Router
  - *Kanten*: Übertragungsabschnitte (Kantenkosten z.B. Verzögerung, Preis,…)
- **Pfad**: Folge von Knoten (meist Pfad mit kürzester Länge gesucht)

Routing-Verfahren — Dynamik

- **Nicht adaptiv**: Routen ändern sich selten, wesentlich seltener als Verkehr
- **Adaptiv**: Routen ändern sich abhängig von Verkehr und Topologie
  - Routenänderungen periodisch oder reaktiv
  - *Zielkonflikt*: Systeme haben ggf kein Live-Abbild des Netzes, ggf hohe Netzbelastung durch Routing-Informationsaustausch

Routing-Verfahren — statisches Routing

Beispiel: Abhängig von Zufallszahl weiterleiten nach B, C oder D

Routing-Verfahren — zentralisiert

- Adaptives Verfahren
- **Routing Control Center**: Für Berechnung/Verteilung der optimalen Pfade → Systeme senden periodisch Zustand an RCC (aktive Nachbarn, Warteschlangenlängen,…)
- **Vorteile**:
  - RCC hat alle Informationen ⇝ kann perfekte Entscheidungen treffen
  - Systeme müssen kein Routing betreiben
- **Nachteile**:
  - Berechnungsdauer für große Netze ggf sehr lang
  - Ausfall RCC lähmt ganzes Netz
  - Inkonsistenzen möglich, da RCC-nahe Systeme Tabellen schneller erhalten
  - starke Belastung des RCC

Routing-Verfahren — isoliert

- **Prinzip**: Jedes System entscheidet nur anhand eigener Informationen
- kein Austausch von Routinginformationen zwischen Systemen
- **Fluten**: Jedes eingehende Datagramm auf jeder Übertragungsleitung weiterleiten
- *Fluteindämmung*:
  - *Sequenznummern* für Duplikaterkennung
  - *Lebensdauerkontrolle* durch Zählen der Übertragungsabschnitte (*hops*)
- *Varianten*:
  - *selektives Fluten*: Weiterleitung nicht auf allen Übertragungsabschnitten
  - *random walk*: Zufällige Auswahl eines Übertragungsabschnittes
- **Hot Potato**: Datagramme so schnell wie möglich weiterleiten ⇝ Übertragungsabschnitt mit kürzester Warteschlange wählen
- *Varianten*:
  - nie an Herkunftsleitung weiterleiten
  - Kombination mit statischem Routing: statisches Verfahren zur Auswahl von Leitung mit Warteschlangenlänge unter Schwellwert

Routing-Verfahren — Verteiltes adaptives Routing

- **Prinzip**: Systeme tauschen Routing-Informationen mit Nachbarn aus, jedes System unterhält Routing-Tabelle
- **Varianten**:
  - periodischer Informationsaustausch
  - Austausch nur bei größeren Änderungen

Routing-Algorithmen — Übersicht

- **Distanz-Vektor-Algorithmen**:
  - Distanz als Routing-Metrik
  - jeder Router kennt Distanz zu allen anderen Systemen im Netz
  - dazu Austausch der Distanzen zwischen Nachbarn
  - *Problem*: kürzerer langsamer Weg wird längerem schnelleren vorgezogen
- **Link-State-Algorithmen**:
  - Unterschiedliche Routing-Metriken
  - berücksichtigt aktuelle Zustände der Netzanschlüsse
  - jeder Router kennt und nutzt ganze Netztopologie zur Berechnung

Link-State vs. Distanz-Vektor

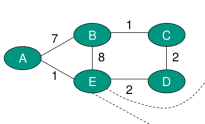
- **Komplexität Kontroll-Pakete**:
  - *Link-State*: jedes System muss Kosten aller Links kennen, Änderungen müssen an alle Systeme geschickt werden ⇝ $O(nE)$ Pakete
  - *Distanz-Vektor*: Änderungen nur an benachbarte Systeme weitergegeben
- **Konvergenzgeschwindigkeit**:
  - *Link-State*: schnelle Konvergenz, schleifenfrei, Oszillationen möglich
  - *Distanz-Vektor*: langsame Konvergenz, Schleifen + Count-to-∞ möglich
- **Robustheit**:
  - *Link-State*: Routenberechnungen separiert ⇝ Robustheit
  - *Distanz-Vektor*: ein System kann inkorrekte Pfade zu allen Zielen verbreiten
- **Fazit**:
  - *Link-State*: Konvergiert schneller, ist robuster
  - *Distanz-Vektor*: einfacher zu implementieren

Routing-Algorithmen — Distanz-Vektor

- **verteilt**: jeder Router erhält Infos von direkten Nachbarn, führt Berechnung durch und verteilt dann neue Informationen an direkte Nachbarn
- **iterativ**: Verteilen + Berechnen von Informationen so lange, bis keine Information mehr ausgetauscht wird

Distanz-Vektor — Distanz-Vektor-Tabelle

- **Distanz-Vektor-Tabelle**:
  - Grundlegende, in jedem System vorhandene Datenstruktur
  - Zeilen für alle möglichen Ziele, Spalten für direkte Nachbarn
- **Beispiel**: X will Daten über direkten Nachbar Z an Y weiterleiten ⇝ $D^X(Y, Z) = c(X, Z) + \min_w\{D^Z(Y, w)\}$
- **Beispiel**: $D^E(A, D)$
  - erster Übertragungsabschnitt: $E \to D$
  - Tabelleneintrag: Kosten $E \to D$ (2) + minimale Kosten $D \to A$ (3)



Ziel \ Next Hop	A	B	D
A	1	14	5
B	7	8	5
C	6	9	4
D	4	11	2

Distanz-Vektor — Distanz-Vektor-Algorithmus

- von Bellman und Ford
- **Initialisierung**:
  - für alle Nachbarn v: $D^X(*, v) = \infty$, $D^X(v, v) = c(X, v)$
  - für alle Ziele y: sende $\min_w D^X(y, w)$ allen Nachbarn (w enthält Nachbarn)
- **Schleife**:
  - geänderte Abschnittskosten: für alle Ziele y: $D^X(y, V) := D^X(y, V) + d$

- Update von Nachbarn: kürzester Pfad von V zu Ziel Y um α geändert
  $\rightsquigarrow D^X(Y, V) := c(X, V) + \alpha$
  $\rightsquigarrow$ Falls neuer Minimalwert für ein Ziel Y, sende an alle direkten Nachbarn
- **Komplexität**: $O(n^2k)$ für n Knoten und k Kanten

Distanz-Vektor — Updateausbreitung

- **Good news**: schnelle Ausbreitung
- **Bad news**: langsame Ausbreitung, ggf Routing-Schleifen
  $\rightsquigarrow$ **Count to Infinity-Problem**
- **Poisoned Reverse**: Vermeidung von Routing-Schleifen, indem Routing-Information Y vorenthalten wird, wenn Weg über Y kürzer

Routing-Algorithmen — Link-State-Routing

- **Prinzip**: Jedes System berechnet kürzeste Pfade durch gesamtes Netz
  - Systeme müssen zu Beginn nur direkte Nachbarn kennen
  - Entdecken von neuen Nachbarn zB mittels `HELLO`-Pakete
  - *link state broadcast*: Identität + Kosten zu Nachbarn werden allen Routern im Netz weitergeleitet (Fluten)
  - Systeme lernen Topologie durch LSBs der anderen Systeme
  - *Ergebnis*: Alle Systeme haben *identisches* Wissen über Netz
- **Implementierung**: Dijkstra-Algorithmus

software defined networking

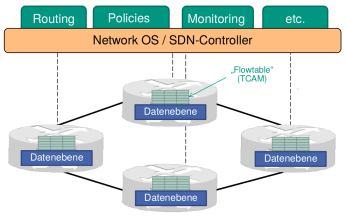
- **Zentrale Eigenschaften**:
  - Separierung von Kontroll- und Datenebene
  - Flow-basierte Paketweiterleitung
  - Logik an externen Controller ausgelagert
  - Netzwerk programmierbar
- **Umsetzung**: *open flow*-Protokoll (quasi-Standard, Alternativen existieren)
  → regelt Kommunikation zwischen Controller und Switch

Traditionelles IP-Routing

- Kontroll- und Datenebene in jedem Router
- **Vorteile**:
  - Ausfallsicherheit (selbstorganisiert, verteilte Kontrolle, hohe Redundanz)
  - Schnelle Reaktion (optim Routing-Hardware, lokale Routingentscheidung)
  - Bewährtes Konzept
- **Nachteile**:
  - proprietäre Management-Schnittstellen (Mischbetrieb schwierig)
  - unflexibel (neue Funktionen hzfg schwierig, aufwändige Standardisierung)
  - teuer (hochqualifiziertes Personal + Overprovisioning benötigt)

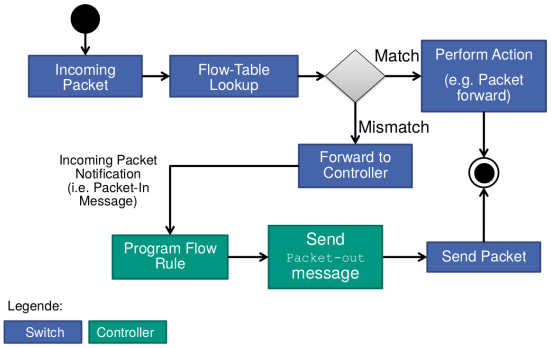
SDN-Routing

- **Vorteile**:
  - logisch zentralisierte Sicht (Controller hat Netzüberblick, einfacher Einsatz von Graphenalgorithmen)
  - neue Funktionalität in Software (als App im Controller, kürzere Entwicklungszeit)
  - Trennung von Kontroll- und Datenebene (Innovationen unabhängig möglich, herstellerunabhängig durch offene Schnittstellen)
- **Nachteile**:
  - Skalierbarkeit
  - *single point of failure*
  - Kommunikation mit Controller langsam



SDN — Paketweiterleitung

- **Traditioneller IP-Router**: kennt keine Flows, Weiterleitung über Ziel-IP-Adresse (Longest Prefix Matching)
- **SDN-Switch**: Weiterleitung über Flowtable, nutzt verschiedene IP-Kopf-Felder, speichert Zustand pro Flow



Sicherungsschicht

Terminologie

- **Knoten**: Endsysteme + Router
- **Links**: Übertragungsabschnitt zwischen benachbarten Knoten
- **Rahmen**: Pakete auf Schicht 2 (IP-Datagramme eingekapselt)
- **Aufgabe**: Übertragung von Datagrammen zw Nachbarknoten über Link

Aufgaben

- **Strukturierung** des Datenstroms (*framing*)
  $\rightsquigarrow$ Datagramm in Rahmen einkapseln hinzufügen
- **Medienzugangskontrolle** bei geteilten Medien
- **Adressierung** mittels MAC-Adressen
- Je nach angebotenem Dienst *Fehlererkennung/-behebung* bzw. *Flusskontrolle*
- **(Semi-) Broadcast**: Alle Stationen sehen alle Rahmen (zB WLAN = semi-broadcast)
- **Punkt-zu-Punkt-Link**: Zwei Stationen sind über dedizierten Link verbunden (zB switch-basiertes Ethernet)
  - *Simplex*: Übertragung in eine Richtung
  - *Halbduplex*: Übertragung in beide Richtungen, nicht zeitgleich
  - *(Voll-) Duplex*: Übertragung in beide Richtungen, zeitgleich

Sicherungsschicht — Implementierung

Sicherungsschicht ist in jedem Knoten (Endsystem, Router, Switch) implementiert (auf Netzadapter oder auf Chip), an Systembus angeschlossen (Kombination von Hardware, Software und Firmware)

Sicherungsschicht — Fehlererkennung

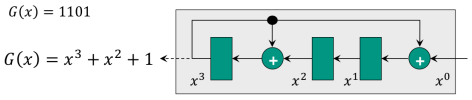
- **Wie Schicht 4**: Erkennung/Behebung von Bit- und Paketfehlern
- **Unterschied Schicht 4**:
  - zu sendende/empfangende Bitfolge wird bitseriell betrachtet
  - Internetprüfsumme basiert auf Wörtern, die bereits im Speicher stehen
- Rahmen erhält Sicherungssequenz *frame check sequence* (FCS, üblicherweise als Anhang am Rahmenende)

Fehlererkennung — cyclic redundancy check (CRC)

- Jede zyklische Verschiebung eines Codeworts führt wieder zu einem Codewort
- **Code → Polynom**: `0101` $\rightarrow 0x^3 + 1x^2 + 0x^1 + 1x^0 = x^2 + 1$
- **Generatorpolynom**: von $G(x)$ generierte Code ist
  $C := \{v(x) \mid \deg(v(x)) < n \wedge G(x) \text{ teilt } v(x)\}$
- **Prinzip**:
  - gleiches Polynom $G(x)$ für Sender und Empfänger
  - *Sender*:
    - m Bit Rahmen $\rightarrow M(x)$ (Polynom/Codewort)
    - hängt $r = \deg(G(x))$ Nullen an Daten ($\rightarrow x^r \cdot M(x)$)
    - berechnet Rest von $M(x)/G(x)$
    - hängt Rest an ursprüngliche Daten (statt der Nullen von oben)
  - *Empfänger*: Dividiert durch $G(x)$
    - Ergebnis 0: keine Fehler erkannt
    - Ergebnis $\neq 0$: Fehler!

CRC — Hardwareimplementierung

- Rückgekoppelte Schieberegister → CRC bei Durchschieben berechnet
- **Prinzip**:
  - Bitweises Empfangen der Daten, durchlaufen Schieberegister
  - Rückkopplung durch `XOR`-Gatter an 1-Stellen des Gens (ohne höchstes Bit)
  - Nach Durchlaufen von Codewort + angehängte 0en Prüfsumme im Register



Multiplexing — Medienzugriff

- **Problem**: Link von mehreren Knoten parallel benutzt
- **Varianten**:
  - feste Mediumszuteilung (eine Dimension, Punkt-zu-Punkt-Verbindungen)
  - konkurrierende Nutzung → Zugriffsorganisation notwendig
- **Dimensionen**: Raum r, Zeit t, Frequenz f, Code c
- **Wichtig**: Schutzabstände erforderlich

Multiplexing — Raum

- Raumeinteilung in Sektoren (zB gerichtete Antennen)
- **Kupfermultiplex**: Zuordnung dedizierter Leitungen
- **Einsatz**: Mobilfunkzellen
- Space Division Multiple Access (SDMA)

Multiplexing — Frequenz

- **Prinzip**: verfügbare Bandbreite wird in Frequenzabschnitte unterteilt
- **Vorteile**:
  - keine dynamische Koordination nötig
  - auch für analoge Signale möglich
- **Nachteile**:
  - Bandbreitenverschwendung bei ungleichmäßiger Auslastung
  - unflexibel
- **Einsatz**: DSL

Multiplexing — Zeit

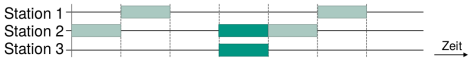
- **Prinzip**: Kanal belegt ganzen Frequenzraum für festgelegte Zeit
- **Vorteile**:
  - nur ein Träger gleichzeitig auf Medium
  - auch bei großer Teilnehmerzahl hoher Durchsatz
- **Nachteil**: genaue Synchronisation nötig
- **Einsatz**: Ethernet, WLAN
- **Hinweis**: Standard-Multiplexverfahren im Folgenden

Multiplexing — Code

- **Prinzip**: alle Stationen zur gleichen Zeit auf gleicher Frequenz
  - *Sender*: verknüpft Signal mit eindeutiger Pseudozufallszahl
  - *Empfänger*: kann mithilfe bekannter Pseudozufallszahlfolge + Korrelationsfunktion Originalsignal wiederherstellen
- **Vorteile**:
  - keine Frequenzplanung erforderlich
  - großer Coderaum im Vergleich zu Frequenzraum
  - Vorwärtskorrektur + Verschlüsselung leicht integrierbar
- **Nachteile**:
  - höhere Komplexität wegen Signalregenerierung
  - alle Signale müssen bei Empfänger gleich stark ankommen
- **Einsatz**: UMTS

Zeitmultiplex — Zufallsstrategien

- **Aloha**: zufällige, unabh, seltenes Senden; gleichzeitiges Senden = Kollision
- **Slotted Aloha**: Verbesserung von Aloha, Erfordert Knotensynchronisation



- **CSMA** (*carrier sense multiple access*):
  - *Prinzip*: Andere nicht unterbrechen während sie reden
  - *listen before talk*: System prüft vor Senden, ob Medium frei ist
  - *Medium belegt*: später erneut versuchen
  - *Medium frei*: Senden
  - *Kollisionen*, wenn mehrere Systeme gleichzeitig zu Senden beginnen
- **CSMA/CD** (*CSMA with collision detection*)
  - *listen while talk*: Kollisionserkennung durch Abhören während des Sendens
  - *Kollision*: Sendungsabbruch, später neu versuchen

Zeitmultiplex — Umsetzung Ethernet

- **Kollision**:
  1. Sendungsabbruch
  2. Sender sendet *Jamming-Signal*
  3. *Backoff-Algorithmus* regelt Sendungswiederholung
- **Vorraussetzungen**:
  - Senden der Rahmen darf nach Signallaufzeit durch Medium und zurück noch nicht fertig sein
  - Mindestlänge für Rahmen (abhängig von Netzausdehnung + Ausbreitungsgeschwindigkeit) erforderlich
  - zu kleiner Rahmen: Auffüllen auf Mindestlänge (*padding*)

Kollisionsfreier Zugriff — Prinzip

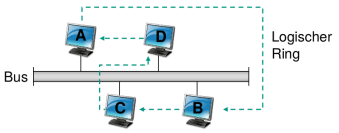
- **Polling**: Kontrolle durch zentralen Knoten
  - Senderecht sequentiell zugewiesen
  - *Nachteil*: koordinierender Knoten nötig, kann ausfallen
  - *Einsatz*: Bluetooth
- **Token Passing**: Senderechtsweitergabe von Knoten zu Knoten
  - *Nachteil*: Knoten können ausfallen → Zugriff blockiert
  - *Einsatz*: Token Ring

Kollisionsfreier Zugriff — Token Ring

- **Prinzip**:
  - Systeme physikalisch Punkt-zu-Punkt-verbunden zu Ring
  - Jedes System hat *Vorgänger* und *Nachfolger*
  - Senderechtszuteilung durch zirkulierendes Token
  - Sendendes System nimmt Daten auch wieder vom Ring
- Monitor: Endsystem zur Überwachung des Rings, Tokenmanagement (komplex!)
- Strukturierte Verkabelung von Gebäuden, Viele Endsysteme möglich

Kollisionsfreier Zugriff — Token Bus

- **Prinzip**:
  - Verbindet Vorteile von Ethernet und Token Ring
  - Busverkabelung wie bei Ethernet (robust: Ausfall eines Endsystem für Netz egal)
  - *Garantierte Antwortzeiten* durch zirkulierendes Token
- **Aufbau**:
  - Alle Stationen physikalisch durch Bus verbunden
  - Bildung eines *logischen Rings*



Lokale Netze — MAC-Adressen

- Theoretisch weltweit eindeutig
- Jeder Netzadapter muss in einem lokalen Netz eindeutige MAC-Adresse haben
- **Funktion**: lokal genutzt, um Rahmen von Interface zu benachbartem, physikalisch verbundenem Interface zu übertragen
- **Format**:
  - 48 Bit (24 Bit von IEEE an Hersteller zugewiesen, 24 Bit durchnummeriert)
  - stehen im NIC-ROM, können aber auch per Software gesetzt werden
  - Darstellung meist hexadezimal (zB `24-2F-EA-76-CC-28`)
  - Broadcast: `FF-FF-FF-FF-FF-FF`

Lokale Netze — address resolution protocol (ARP)

- **Problem**: Welche MAC-Adresse hat nächstes System im eigenen Subnetz?
- **Aufgabe**: MAC-Adresse zu bekannter IP-Adresse ermitteln
- **Prinzip**: dynamisch Adresszuordnungen lernen
- **ARP-Cache**: kleine Tabelle auf jedem System, Einträge bei Bedarf gelernt; Eintrag IP + MAC + maximale Lebenszeit (typisch 20 Minuten)

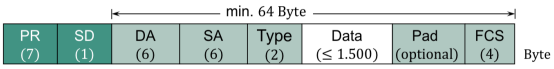
ARP — Adressauflösung

- **Szenario 1**: A sendet Datagramm an B in selbem Subnetz
  - *Fall 1*: ARP-Cache von A hat Eintrag für B → Paket verschicken, Timeout neu setzen
  - *Fall 2*: ARP-Cache von A hat Eintrag für B *nicht* → Broadcast *ARP-Request* mit IP von B, Jeder Knoten liest ARP-Request (ARP-Reply falls eigene IP), A trägt Infos in ARP-Cache ein
- **Szenario 2**: A sendet Datagramm an B in anderem Subnetz
  1. A sendet ARP-Request für Router R

2. A sendet Datagramm an IP von B und MAC von R
3. Router empfängt Datagramm, setzt Ziel-MAC auf B, Sender-MAC auf R
4. Router leitet Datagramm weiter

Lokale Netze — Ethernet (IEEE 802.3)

- **Medienzuteilung**:
  - zeitmultiplex, variabel, zufälliger Zugriff, CSMA/CD
  - Kanal wird logisch in Zeitschlitze fester Länge aufgeteilt
  - Dauer = minimale Rahmenlänge, Kollisionserkennung vor Zeitschlitz-Ende
  - Exponentieller Backoff: Warte nach Kollision i zufällig $[0, 2^i - 1]$ Zeitschl
- **Netztopologie**: Ursprünglich Bus-, heute Sterntopologie (Switches statt Repeatern)
- **Varianten**: [Datenrate][Baseband/Broadband][Medium] (z.B. 10Base5)
- Ethernet-Rahmen (immer gleich)



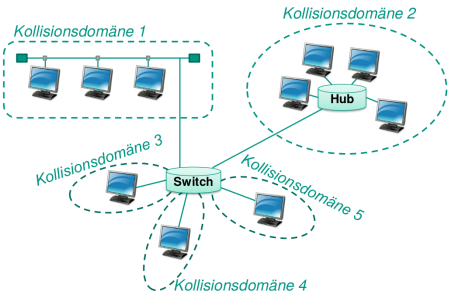
Präambel, Start of Frame Delimiter, Destination Address, Source Address, Type/Length, Data, Padding, Frame Check Sequence

Ethernet — Switches

- **Prinzip**: Schicht-2-Netzkopplung (innerhalb *eines* IP-Subnetzes)
  - Leitet Rahmen zwischen Interfaces weiter und puffert sie zwischen
  - Trennung von Inter- und Intranetz-Verkehr → Erhöhung Netzkapazität
  - Switches nicht sichtbar für Endsysteme
- **Ziel**: Selbstorganisierte Netzkonfiguration mit Switches
- **Aufgaben**:
  - Schleifenfreie Netztopologie (*spanning tree* Algorithmus)
  - Wege zwischen Endsystemen (selbstlernend; Ziel unbekannt: Fluten)

Ethernet — Kollisionsdomänen

Netzbereich, der von Kollision betroffen ist (gemeinsames Broadcastmedium)



virtual local area network (VLAN)

- **Idee**: Logische Trennung von Datenverkehr auf Ethernet-Ebene
  ⇝ virtuelle Leitung
- **Sicherheit**: Trennung in logische Medien → gezielte Systemgruppierung + bessere Netzstruktur-Kontrolle
- **Flexibilität**:
  - Einfache Reorganisation der logischen Medien möglich
  - keine Änderungen an physikalischem Medium (Neuverkabelung) nötig
- **Performance**: Broadcast-Last eines Netzes sinkt, wenn physikalisches Medium in mehrere logische aufgeteilt wird

VLAN — Interface-basiert

- Ein einziger physikalischer Switch arbeitet als mehrere virtuelle Switches
- Jeweils mehrere Interfaces werden zu einem virtuellen Switch gruppiert
- **Verkehrsisolation**: Rahmen von einem Interface können nur Interfaces in der gleichen Gruppe erreichen ⇝ Sicherheit, Performance
- **Flexible Zuweisung**: Interfaces dynamisch anderen VLANs zuordnen
- **Weiterleitung** zwischen VLANs über Routing
- **Trunks**: Transport von Rahmen zwischen multi-switch-VLANs
  - *VLAN-ID*: Jedes VLAN erhält Kennzeichner
  - Ethernet-Frames werden mit VLAN-ID getaggt
  - Switches entfernen Tagging vor Auslieferung an Endsystem

Architektur

Grundmodell

- Daten überbrücken räumliche Distanz (abstrakten Übertragungsabschnitt)
- Abstraktion auf Basis von **Schichten**, stellen Dienste über Schnittstellen bereit
- Höhere Schicht erfordert Dienste der darunterliegenden Schicht
- **Ziele**:
  - Komplexitätsreduktion (Vereinheitlichung, Modularisierung)
  - Interoperabilität (Hersteller-/Systemunabhängigkeit)
  - Flexibilität und Erweiterbarkeit
- **Horizontale Kommunikation**: zwischen Sender und Empfänger, Protokollinstanzen einer Schicht tauschen Daten aus um Dienst zu erbringen
- **Vertikale Kommunikation**: zwischen verschiedenen Schichten in einem System, Protokollinstanz Schicht n greift auf Dienste von Protokollinstanz Schicht n-1 zu

OSI-Referenzmodell

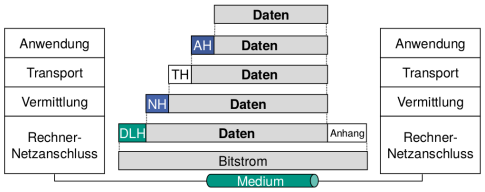
- **Logisches** Modell, nicht Implementierungsmodell
- Keine Protokolle (nur Prinzipien), offener Standard
- Unterteilung in Transportsystem (1-4) und Anwendungssystem (5-7)
- **Schicht 1**: Bit-Übertragungsschicht (*physical layer*)
  - *Ziel*: Übertragungsqualität
  - Bitübertragung
  - Verwendung von Leitungscodes usw
  - *keine* Pufferung, *kein* zuverlässiger Dienst
- **Schicht 2**: Sicherungsschicht (*data link layer*)
  - *Ziel*: Kommunikation zwischen physikalisch benachbarten Systemen
  - Erkennung/Behebung von Fehlern der Bitübertragungsschicht
  - Bitstrom in Rahmen gliedern
  - Puffern bei Sender + Empfänger
- **Schicht 3**: Vermittlungsschicht (*network layer*)
  - *Ziel*: Verknüpfung von Übertragungsabschnitten zu Netz
  - Wegewahl im Kommunikationssystem
  - Geräteadressierung, Multiplexing
- **Schicht 4**: Transportschicht (*transport layer*)
  - *Ziel*: Übertragung von Daten zwischen Anwendungen
  - Abstrahiert von Diensten der Vermittlungsschicht
  - Fehlererkennung/-behebung
  - Pufferung, Multiplexing
  - Adressierung von Transportdienstnutzern
- **Schicht 5**: Sitzungsschicht (*session layer*)
  - *Ziel*: Nichtunterbrechbarkeit von Kommunikationsbeziehungen
  - Datenaustausch-Gliederung nach Gesichtspunkten der Anwendung
  - Ablaufsteuerung/Koordination
  - Bereitstellen von Sitzungen
- **Schicht 6**: Darstellungsschicht (*presentation layer*)
  - *Ziel*: Einheitliche Datendarstellung
  - Kommunikation zwischen heterogenen Geräten
  - Beibehaltung der Informationssemantik bei Überführung der Syntax
- **Schicht 7**: Anwendungsschicht (*application layer*)
  - *Ziel*: Austausch anwendungsabhängiger Daten

Internet-Referenzmodell

- Einfacheres Modell, nur 4 Schichten (manchmal 5 bei Trennung von Sicherung und Bit-Übertragung)
- Darstellungs- und Sitzungsaufgaben in Anwendungen verlagert

Datenkapselung

- Information wird durch alle Schichten durchgereicht
- Daten werden in jeder Schicht gekapselt (mit Header und/oder Trailer versehen)



Protokolle

Regeln und Formate für Datenaustausch *innerhalb* einer Schicht

Dienste

- Bündelung zusammengehöriger Funktionen
- Zusammenwirkung von Protokollinstanzen *innerhalb* einer Schicht
- Schichten gehen über gesamtes Kommunikationssystem hinweg
- *Dienstfunktion*: einzelne Dienstteile unabhängig voneinander nutzbar
- *Dienstprimitiv*: Einzelvorgänge einer Dienstfunktion
  - *request* (Req): Beauftragung (Nehmer → Geber)
  - *indication* (Ind): Partnerbenachrichtigung (Nehmer ← Geber)
  - *response* (Rsp): Partnerbeantwortung (Nehmer → Geber)
  - *confirmation* (Cnf): Abschlussbenachrichtigung (Nehmer ← Geber)
- *Dienstehierarchie*: Dienst baut auf anderen Diensten auf (Dienstbringer/-nehmer)
- *Dienstzugangspunkte*: Dienstschnittstellen

Ablauf — Webseitenaufruf

- **Start**: Einstecken Netzwerkkabel
- **Ende**: Seitenempfang
- **Netzverbindung**: IP erhalten, Router + DNS kennenlernen
  1. DHCP-Anfrage (verpackt in UDP, IP, 802.3)
  2. Ethernet-Paket wird im LAN gebroadcastet
  3. DHCP-Server im Router empfängt + entpackt Paket
  4. DHCP-Server erstellt DHCP ACK-Paket mit Client-IP, Router-IP, DNS-IP
  5. DHCP-Antwort wird verpackt und direkt an Client gesendet
  6. Client empfängt und entpackt Paket
- **ARP**: MAC des Routers kennenlernen
  1. ARP-Anfrage an Broadcast-Adresse
  2. Router sendet seine MAC in ARP-Antwort
- **DNS**: IP-Adresse der angeforderten Webseite kennenlernen
  1. IP-Datagramm mit DNS-Anfrage wird von LAN-Switch zu lokalem Router geleitet
  2. IP-Datagramm: lokales Netz → ISP-Netz → DNS-Server (mit RIP oder OSPF)
  3. Paket wird an DNS-Server entpackt
  4. DNS-Server antwortet Client mit angeforderter IP
- **TCP**: Aufbau einer TCP-Verbindung
  1. Eröffnung eines TCP-Sockets zum Webserver
  2. TCP-SYN-Segment wird zu Server geroutet
  3. Server antwortet mit SYNACK
- **HTTP**: Webseite laden
  1. HTTP-Anfrage wird per TCP-Socket gesendet
  2. IP-Datagramm mit HTTP-Anfrage wird zu Webserver geroutet
  3. Server antwortet mit HTTP-Antwort
  4. IP-Datagramm mit HTTP-Antwortet wird zurück zu Client geroutet

Sicherheit

Angriffe

- Klassischer Angreifer (Dolev-Yao): Omnipräsent im Netz
  - kann Pakete abhören / manipulieren / eigene Pakete erzeugen
  - kein Zugriff auf Endsysteme
  - keine Entschlüsselung ohne Schlüssel
- Abhören, Einfügen, Manipulieren, Man in the Middle, Replay, DoS,

⇝ **System/Protokoll** verwendet **Bausteine** um **Schutzziele** zu realisieren und vor **Angriffen** zu schützen

Schutzziele (CIA)

- *Anforderungen an eine Komponente oder ein System, die erfüllt werden müssen, um schützenswerte Güter vor Bedrohen zu schützen*
- **Confidentiality** (Vertraulichkeit): keine unautorisierte Informationsgewinnung (Bausteine: Asymmetrische/Symmetrische Verschlüsselung)
- **Integrity** (Integrität): Kein Ersetzen, Einfügen oder Löschen von Daten
  - *starke Integrität*: Keine unautorisierte Manipulation von Daten möglich
  - *schwache Integrität*: Manipulation von Daten nicht *unbemerkt* möglich
  - Manipulationen in vielen Fällen nicht zu verhindern ⇝ schwache Integrität
  - Bausteine: Tamper Proof-Module, Message Authentication Codes (MAC)
- **Availability** (Verfügbarkeit): beschreibt, in welchem Maße die Systemfunktionalität von berechtigen Subjekten unabhängig von Einflüssen in Anspruch genommen werden kann
- **Authentizität**: angegebene Datenquelle ist tatsächl Quelle + Datenintegrität
  - *Subjektechtheit*: Bob will sicherstellen, dass er tatsächlich mit Alice spricht ⇝ *Authentifikation*
  - *Datenechtheit*: Bob will sicherstellen, dass Daten tatsächlich von Alice sind
  - Bausteine: Zertifikate, Signaturen, gemeinsames Geheimnis
- weitere Schutzziele: Privatheit, (Nicht-)Abstreitbarkeit

Verschlüsselung

- **symmetrisch**: Ent- und Verschlüsseln mit einem Schlüssel, sehr effizient
- **asymmetrisch**: Verschlüsseln: öffentlicher Schlüssel, Entschlüsseln: privater → RSA

Kryptografische Hashfunktion

- Einwegfunktion ($H(m)$ effizient, $H^{-1}(c)$ nicht)
- Zu gegebenen b schwierig, a zu finden mit $H(a) = b$
- Schwache Kollisionsresistenz: Zu a ein a' mit $H(a) = H(a')$ schwer findbar
- Starke Kollisionsresistenz: Paar $a \neq a'$ mit $H(a) = H(a')$ schwer findbar

Integritätssicherung

- Daten sollen beim Empfänger genau so eintreffen, wie sie versendet wurden
- Manipulationen können nicht verhindert, nur erkannt werden -> Schwache Integrität
- **Message Authentication Code**:
  - *Ziel*: Empfänger erkennt Manipulation an empfangenen Daten
  - *Voraussetzung*: Alice und Bob haben gleichen symmetrischen Schlüssel
  - *Vorgehensweise*: Alice hängt Hash v. (Nachricht+Schlüssel) an Nachricht an
- **Digitale Signatur**: Sichert Integrität
  - *Ziel*: Bob kann prüfen, dass wirklich Alice Dokument unterschrieben hat
  - *Vorgehensweise*: Hash des Dokuments mit privatem Schlüssel verschlüsseln, als Signatur mitsenden, entschlüsseln mit öffentlichem Schlüssel
- **Digitales Zertifikat**: Sichert Authentizität
  - Ziel: Verifizieren, dass jemand der ist, für den er sich ausgibt (öffentlicher Schlüssel tatsächlich zu ihm gehört)
  - Problem: kann man nicht selbst überprüfen ⇝ verlassen auf Dritte
  - Vorgehensweise: Überprüfung durch *certificate authority* (CA), ID-Zertifikat = Authentifikation des öffentlichen Schlüssels

E-Mail Sicherheit — PGP (Pretty Good Privacy)

- SSL/TLS nur scheinbar sicher: Backend-Weiterverarbeitung unverschlüsselt
- E-Mail-Verschlüsselung PGP: Bietet Vertraulichkeit, Integrität, Authentizität
- Verwendet symmetrischen Schlüssel, der asymmetrisch Verschlüsselt zusammen mit Nachricht versendet wird
- **Problem**: Öffentliche Schlüssel müssen vor Versand bekannt sein
- Zugehörigkeit zur Mailadresse muss überprüfbar sein
- Bekanntgabe auf öffentlichen Schlüsselservern
- **Web of Trust**: Dezentraler, anarchischer Vertrauensansatz (ohne CAs)
  - Transitive Überprüfung der Authentizität eines öffentlichen Schlüssels
  - Nutzer bestimmt Vertrauen in Signaturen anderer Nutzer (Signatory Trust)
  - Key Legitimacy bestimmt sich aus Anzahl der vertrauten Signaturen
- In der Realität: Viele Probleme (Inseln)

Infrastruktursicherheit

- **Firewall**: Zugriffskontrolle durch Paketfilterung
  - Teile des Netzes vor Eindringen unerwünschter Pakete schützen
  - Netzbereiche (Intern, Internet, Demilitarized Zone) voneinander isolieren
  - Zustandslos (IP, Port, Protokoll, Interface) mit Access Control List (ACL)
  - Zustandsbehaftet (Überwacht TCP-Verbindungen)
  - Application Layer Gateway: Filtern auf Basis von Nutzdaten (z.B. Username)
- **Intrusion Detection and Prevention**:
  - Bekannte Angriffe erkennen / verhindern
  - Deep Packet Inspection (Analyse der Nutzdaten)
  - Anomaly Detection (Alarm bei Abweichungen vom Normalverhalten)
- **Organisatorische Maßnahmen**: Schulungen, Verantwortlicher, Notfallplan, Richtlinien, Datensicherung, …