I. 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

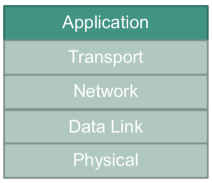
Pakete: Voneinander unabhängige Einheiten für die Weiterleitung – werden durch das Netz zur Zielanwendung geleitet

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)
⇒ 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

II. Anwendungsschicht



Historie

70er/80er: Textbasierte Anwendungen (EMail, Remote Access)
90er: World Wide Web, Instant Messaging, P2P-Filesharing
seit 2000: steigende Vielfalt + Allgegenwärtigkeit: (⇒ 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

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 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, dynamische 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

Selbst-skalierend: Neue Peers erhöhen Kapazität, fordern aber auch Dienste an

Web und HTTP — Web-Dokumente

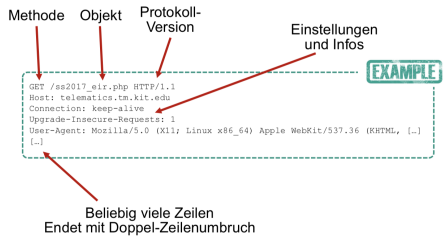
Webseiten bestehen aus Basis-HTML-Datei und anderen Objekten (.js, .png,…)
Jedes Objekt über URL (*uniform resource locator*) referenzierbar

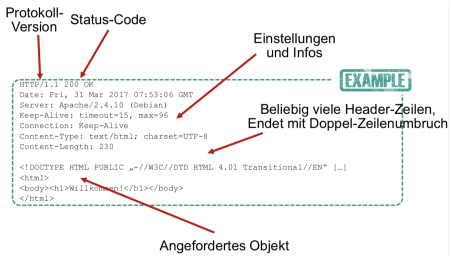
HTTP (Hypertext Transfer Protocol)

ASCII-basiertes Transfeprotokoll 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 dem 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 wurde 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 erfodern mehrere TCP-Verbindungen (evtl. parallel)
Antwortzeit: $2 * \text{RTT} + t_s$ pro Objekt
- eine RTT für Verbindungsaufbau
- eine RTT für HTTP-Anfrage und erste Antwortbytes
- Zeit 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 abhängig von Nutzeridentifikation 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, viel über Nutzer lernen

Mail — Komponenten

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

Mail — SMTP (Simple Mail Transfer Protocol)

Transfer von Mails zwischen Mailservern sowie vom User Agent zum Mailserver
Drei Phasen:
1. Handshake
2. Nachrichtenübermittlung (E-Mail aus Header + Body)
3. 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-Protokoll
Dezentral, ähnlich wie E-Mail
Whatsapp nutzt einen 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 . Top-Level-Domain)
Funktionalitäten:
- *Registrierung* von Namen + IP-Adressen
- *Auflösung* von Namen in IP-Adressen
- *Host Alias*: Löse einfacher zu merkende Alias-Namen in kanonische Namen auf
- *Mail Server Alias*: Liefere E-Mail-Server zu einer Domain
- *Lastverteilung*: Mehrere IPs von redundanten Servern in zufälliger Reihenfolge

Protokoll der Anwendungsschicht, über UDP (Port 53) 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 Name-Server: Erste Anfrage immer zu lokalem Server, Antwort aus eigener Zuordnungsdatenbank, dem Cache oder nach Befragung anderer DNS-Server
Autoritativer Name-Server: Enthält autoritative Abbildungen, jeder Host ist bei einem registriert (in seinem Netz)
Top-Level Domain (TLD) Server
Root-Server: Enthalten nur Einträge für TLDs, fixe IP-Adressen, 13 Root-Server-Cluster

DNS — Anfragen

Rekursiv: kennt angefragter Server Antwort nicht, fragt dieser weitere Server, bis er Antwort zurückliefern kann
Iterativ: kennt ein Server die Antwort nicht, verweist er den *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)
Zwei 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

III. Transportschicht

Internet-Protokollstack

Anwendungsschicht
Transportschicht
Vermittlungsschicht
Sicherungsschicht
Physikalische Schicht

Transportschicht — Ziel

Verbergen von Transportdetails vor höheren Schichten
- Fehlercharakteristika
- genutzte Technologien
- …

Bereitstellung von Transportdiensten
⇝ **Nutzer-zu-Nutzer-Kommunikation**

Transportprotokolle — Prinzip

Transportprotokoll läuft auf Endsystemen
Sender:
- *Segmentieren* von Anwendungsnachrichten
- *Weiterleiten* an Vermittlungsschicht

Empfänger:
- *Reassemblieren* der Segmente in Nachrichten
- *Weiterleiten* an Anwendungsschicht

Transportschicht — Transportdienste

UDP (*user datagram protocol*):
bietet verbindungslosen, **unzuverlässigen** Transportdienst
TCP (*transmission control protocol*):
bietet verbindungsorientierten, **zuverlässigen** Transportdienst

Transportdienst — unzuverlässig vs zuverlässig

Unzuverlässig:
- unklar, wieviel der gesendeten Daten heil ankommt
- keine Fehlermaßnahmen

Zuverlässig:
- *Korrektheit*, *Vollständigkeit*, *Reihenfolge* garantiert richtig
- keine Duplikate
- keine Phantom-Pakete
- Fehlermaßnahmen existieren

Schicht vs Dienst

Schicht: Abstraktion

Dienst: Bündelung zusammengehöriger Funktionen
- Höhere Schicht nutzt Dienst darunterliegender Schicht
- Dienste werden an Dienstzugangspunkt einer Schicht bereitgestellt

Port

= *Adressen der Transportschicht*
Unstrukturierte Nummer (16 Bit), 0 bis 65535
Well known ports: viele Portnummern unter 1024 für häufig benutzte Anwendungen (Telnet, HTTP, …) reserviert

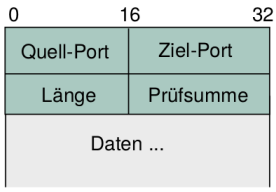
IP-Adressen

= *Adressen der Vermittlungsschicht*
IPv4: 32 Bit (z.B. 207.142.131.235)
IPv6: 128 Bit (z.B. 2001:0db8:85a3:08d3:1319:8a2e:0370:7344)
⇝ Internetweite Adressierung eines Anwendungsprozesses: IP-Adresse + Port

UDP

RFC768 — *sehr einfaches Transportprotokoll mit sehr geringem Overhead*
Eigenschaften:
- (De-) Multiplexen von Segmenten für Prozesse
- *Prüfsumme* für Bitfehler
- *verbindungslos*
- *best effort*: keine Zusagen über Auslieferung bei Empfänger
- *Unreguliertes Senden*: kann Daten so schnell senden wie von Anwendung geliefert und von Netz abgenommen
- *keine Verbindungsaufbauphase*: sofortiges Senden ⇝ keine weitere Verzögerung
- *kein Verbindungszustand*: keine Verbindungsinformationen im Endsystem ⇝ skaliert z.B. für Server besser

Verwendung:
- **Multimedia**
- **DNS**



Protokollmechanismen

Ziel: *Datenaustausch zwischen Anwendungen/Geräten ermöglichen*
⇝ Festlegen von Formaten + Regeln für Datenaustausch nötig
Problem: Fehler bei Datenübertragung möglich

Bitfehler

Verfälschung von Bits während dem Datentransport
Ursachen:
- Dämpfung Übertragungssignal
- Übersprechen
- Synchronisationsverlust
- …

Einzelbitfehler: ein *einzelnes* Bit fehlerhaft
Bündelfehler: mehrere aufeinanderfolgende Bits fehlerhaft

Paketfehler

Fehlerarten:
- Verlust
- Duplizierung
- Phantom-Paket
- Reihenfolgenvertauschung

Fehlerursachen:
- Zwischennetzüberlastung
- Unterschiedliche Wege durch Netz
- Verfrühte Datenwiederholung
- …

Fehler — Häufigkeit und Auswirkungen

Bitfehlerrate: Maß für Fehlerhäufigkeit
$$\text{Bitfehlerrate} = \frac{\text{Summe gestörter Bits}}{\text{Summe übertragener Bits}}$$
Fehlerauswirkungen: 20ms Störung in
- Telex (50bit/s ⇝ Bitdauer 20ms): 1 Bit fehlerhaft (Einzelbitfehler)
- Gigabit-Ethernet (1Gbit/s ⇝ Bitdauer 1ns): 20MBit fehlerhaft (Bündelfehler)

Fehler — Gegenmassnahmen

Fehlererkennung (*error detecting code*, EDC)
- Redundanz zu Daten hinzufügen
- Ausreichend stark unterschiedliche Codewörter verwenden

Fehlerkorrektur (*forward error correction*, FEC)
- Fehler mittels Redundanz korrigieren

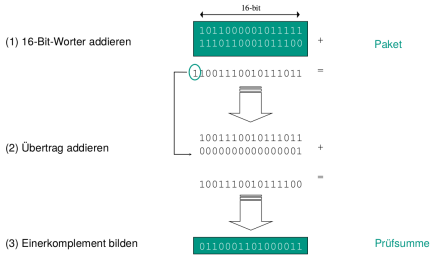
Wiederholungsaufforderung (*automatic repeat request*, ARQ)
- Empfänger teilt Sender Ergebnis der Fehlerkorrektur mit

Fehlerkontrolle — Bitfehler

Problem: Wie Bitfehler erkennen?
Ansatz: Hinzufügen von Redundanz
Paritätsprüfung: bekannt

Bitfehler — Internet-Prüfsumme

Prinzip: Aufaddieren aller übertragenen Wörter (16 Bit, als Integer interpretiert) ⇝ Prüfsumme
Nachteil: Falsche Reihenfolge kann nicht erkannt werden



Bitfehler — UDP-Prüfsumme

Sender:
- UDP-Segment + -Kopf wird als Folge von 16 Bit-Wörtern aufgefasst
- Prüfsumme berechnen und in UDP-Kopf einfügen

Empfänger:
- Prüfsumme des UDP-Segments berechnen - Prüfsummen vergleichen

Fehlerkontrolle — Paketfehler

Erkennung:
- Sequenznummern (*sequence number*)
- Zeitgeber (*timer*)

Behebung:
- Quittungen (*acknowledgements*)
- Sendewiederholungen (*retransmissions*)

Paketfehler — Sequenznummern

Problem: Empfänger weiß nicht, ob Pakete richtig (Reihenfolge, Duplikate, Vollständigkeit) angekommen sind
Prinzip: Pakete werden durchnummeriert

Paketfehler — Quittungen

Problem: Sender erfährt nicht, ob Paket nicht (korrekt) angekommen ist
Prinzip:
- Empfänger informiert Sender über Erhalt ⇝ *Acknowledgement* (ACK)

Varianten:
- *positive Quittung*: Empfänger sagt Sender, dass er Daten erhalten hat (ACK)
- *negative Quittung*: Empfänger sagt Sender, dass er Daten *nicht* erhalten hat (NACK)
- *selektive Quittung*: Quittiert einzelnes Paket — z.B. bei Verlustvermutung (NACK)
- *kumulative Quittung*: Quittiert Paketmenge — z.B. alle Pakete bis bestimmte Sequenznummer sind ok

Paketfehler — Zeitgeber

Problem: Sender merkt nicht, wenn Paket nicht angekommen ist
Prinzip: Durch zeitliche Obergrenze wird *vermutet*, dass Paket nicht angekommen ist ⇝ *Sendewiederholung*

Sendewiederholung — ARQ

= *automatic repeat request*
Grundlegende Sendewiederholungsvariante
Varianten:
- Wann werden Quittungen versendet?
- Wann wird eine Sendung wiederholt?

Sendewiederholung — Stop-and-Wait

= einfaches ARQ-Verfahren
Prinzip:
- Sender wartet auf Quittung für gesendetes Paket
- Erst *nach* Quittungserhalt wird nächstes Paket gesendet
- keine Quittung ⇝ Sendewiederholung
- Wartezeit durch Zeitgeber geregelt

Stop-and-Wait — Sequenznummern

Problem: Empfänger kann Paket doppelt erhalten (nicht erkennbar)
Prinzip: Pakete werden mit Sequenznummern versehen (für Stop-and-Wait reicht 1 Bit)
Auslastung: $U = \frac{1}{1+2a}$ (mit $a = \frac{t_a}{t_s}$)

Bandbreitenverzögerungsprodukt

$=: \frac{m}{v}\,r$ (Länge m, Ausbreitungsgeschwindigkeit v, Datenrate r)
= **Speicherkapazität des Mediums**

Sendewiederholung — Go-Back-N ARQ

Zeil: Leistungsfähigkeit von Stop-and-Wait erhöhen
Prinzip:
- Sender sendet mehrere Pakete bis Quittungspflicht
- begrenzte Anzahl an nicht quittierten Paketen (durch **Fenster** (*window*) begrenzt)
- Quittierung durch kumulative Quittungen

Fehlerfall:
1. Empfänger empfängt fehlerhaftes Paket
2. Empfänger verwirft alle nachfolgenden Pakete 3. Sender wartet auf Ablauf des Zeitgebers
4. Sender wiederholt alle nicht quittierten Pakete

Fragen:
- Wo müssen Pakete gepuffert werden?
- Wieviele Pakete müssen gepuffert werden?

Auslastung: $U = \begin{cases} 1 & W \geq 1+2a \\ \frac{W}{1+2a} & \text{sonst} \end{cases}$

Sendewiederholung — Selective Repeat ARQ

Ziel:
- Auslastung von Stop-and-Wait erhöhen
- Datenaufkommen von Go-Back-N reduzieren

Prinzip: Wie Go-Back-N, Empfänger quittiert selektiv
Fehlerfall:
1. Empfänger bestätigt nachfolgende, korrekt empfangene Pakete
2. Sender wiederholt nur nicht korrekt empfangene Pakete

Fragen:
- Wo müssen Pakete gepuffert werden?
- Wieviele Pakete müssen gepuffert werden?
- Vor-/Nachteile von Go-Back-N und Selective Repeat

Sendewiederholung — Selective Repeat vs. Selective Reject

Selective Repeat:
- Fehlerhaftes Paket wird nicht bestätigt
- Sender wartet auf Timeout

Selective Reject:
- Empfänger versendet für fehlerhaftes 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 - Sendet muss Empfangspuffergröße berücksichtigen

Anforderungen:
- einfach
- möglichst wenig Netzressourcen nutzen
- fair
- stabil

Flusskontrolle — Halt-und-Weiter

Prinzip:
- Empfänger kommt nicht mehr mit ⇝ `halt`-Signal
- Empfänger wieder verfügbar ⇝ `weiter`-Signal

Bewertung:
- nut auf Vollduplex verwendbar
- nicht effektiv bei hohen Verzögerungen
- Probleme bei Verlust der `halt`-Meldung

Beispiele:
- Fast- + Gigabit-Ethernet

Flusskontrolle — Stop-and-Wait

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

Problem:
- Sender kann nicht zwischen Datenverlust und Überlastung unterscheiden

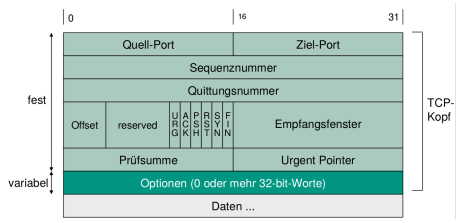
Flusskontrolle — Kreditbasiert

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

TCP — Prinzip

Erhält Bytestrom von Anwendung, übergibt TCP-Segmente an IP
Problem: Wie Bytestrom in TCP-Segmente schnippeln?
Implementierung:
- MSS (*maximum segment size*): Anwendungsdatenlänge (z.B. 1460 Byte)
- Push (PSH in TCP-Segmentkopf): Sender verlangt sofortiges Versenden der Daten
- Zeitgeber: nach inaktivem Zeitintervall werden vorhandene Daten gesendet

Fehlerkontrolle: Sequenznr., Prüfsumme, Quittierungen, Sendewiederholungen
Sequenznummern: pro Byte, nicht pro Segment (erstes Byte in Segment, initiale Sequenznummer von Endsystem *zufällig* gewählt)

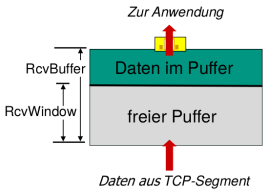


TCP — Felder

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: Wird bei Verbindungsaufbau verwendet, um *connection request* oder *connection confirmation* anzuzeigen
ACK: unterscheidet bei gesetztem SYN-Bit zwischen Request und Confirmation; signalisiert Gültigkeit von Quittungsfeld
FIN: gibt an, dass Sender nichts mehr senden möchte
RST: Verbindung zurücksetzen
PSH: übergebene Daten sollen sofort weitergeleitet werden (idR leer)
Empfangsfenster: für Flusskontrolle
Prüfsumme: Prüfsumme über TCP-Kopf, Daten und Pseudoheader
Urgent-Zeiger: relativer Zeiger auf wichtige Daten
Optionen-Feld: kann Optionen variabler Länge aufnehmen

TCP — Flusskontrolle

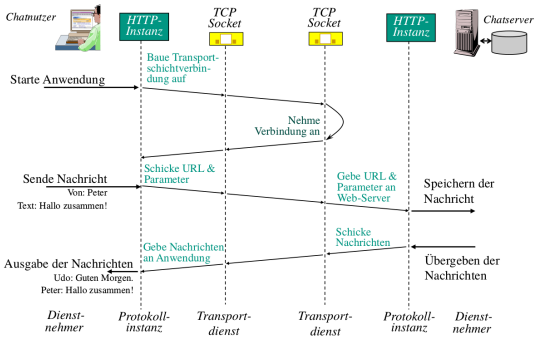
Ziel: Empfängerüberlastung vermeidne
Prinzip:
- *Empfänger*: reserviert Pufferplatz pro Verbindung (explizite Kreditvergabe)
  - `RcvBuffer`: gesamter Pufferplatz (default 4096 Byte)
  - `RcvWindow`: freier Pufferplatz (Empfangsfenster)
  - Sender sendet nicht mehr unbestätigt als in `RcvWindow` passt



Verbindungen — verbindungslos vs. verbindungsorientiert

Verbindungslos:
- Daten werden ohne vorherigen Handshake gesendet
- *Vorteil*: schnelle Datenversendung möglich
- *Nachteil*: kein Feedback, keine Bestätigung

Verbindungsorientiert:
- Verbindungsaufbau vor Datenaustausch, Verbindungsabbau danach
- *Vorteil*: Kommunikationsparameter können ausgehandelt werden
- *Nachteile*: Verzögerter Datenaustausch, Overhead ggf größer als Daten



TCP — Zusammenspiel mit HTTP

Staukontrolle

Ziel: Netzüberlastungssituationen vermeidne
Prinzip:
- Staukontrollfenster (*congestion window*, CWnd) beim Sender beeinflusst maximal zu sendende Datenmenge:
  - $\text{LastByteSent} - \text{LastByteAcked} \leq \min\{\text{CWnd}, \text{RcvWindow}\}$
- Schwellenwert (SSTresh)

Stauerkennung:
- Nutzung von Zeitgebern
- Vermutung einer Stausituation bei ausbleibender Quittung

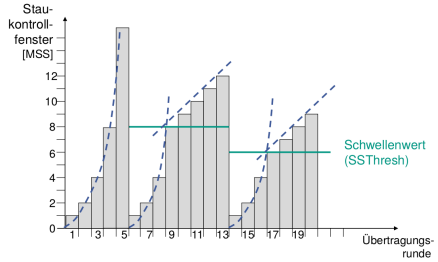
Staubehebung:
- Reduktion von `CWnd`
- Langsames Erhöhen von `CWnd` $\rightsquigarrow$ herantasten an Netzkapazität

Staukontrolle — TCP

Start: `CWnd` = 1 `MSS` (*maximum segment size*)
Slow-Start, falls `CWnd` $\leq$ `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` = $\max(\frac{\text{FlightSize}}{2}, 2 * \text{MSS})$ (FlightSize = unquittierte, gesendete Daten)
- `CWnd` zurücksetzen (neue Slow-Start-Phase): `CWnd` = 1 `MSS`



IV. Vermittlung

Vermittlungstechniken — Leitungsvermittlung

Prinzip: Verbindung erhält *durchgehenden Kanal* mit konstanter Bandbreite für exklusive Nutzung
Multiplexing: *starres Multiplexing* möglich
- *Frequenzmultiplex*: Feste Zuweisung von Übertragungskanal + Frequenzabschnitt
- *Zeitmultiplex*: Feste Zuweisung von Übertragungskanal + Zeitschlitz (*time slot*)

Eigenschaften:
- Aufbau eines durchgehenden Übertragungskanals zwischen Endsystemen
- keine Adressinformation nötig, dafür Zustandshaltung
- zugesicherte, feste Datenrate ⇝ ungenutzte Ressourcen bei Nichtverwendung
- Übertragungsverzögerungen nur physikalisch bedingt
  ⇝ keine Schwankungen durch Puffer
- Reihenfolgentreue Bitfolgenübertragung

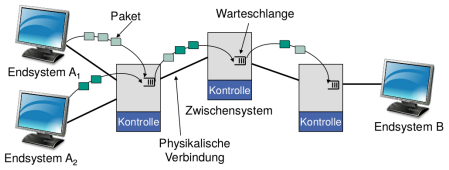
Einsatzgebiete: Telefonnetze, GSM

Vermittlungstechniken — Paketvermittlung

Prinzip: Weiterleitung anhand von Kontrollinformation in Paket
- Zieladresse in Datagrammen
- lokale Kennung bei virtuellen Verbindungen
- Zwischensysteme Speichern Pakete in *Warteschlangen* ⇝ Paketverlust möglich

Eigenschaften:
- Wechselnde Paketwege möglich ⇝ Reihenfolgevertauschung möglich
- Üblicherweise Zeitmultiplex

Varianten:
- *verbindungslos*: Datagramme
- *verbindungsorientiert*: virtuelle Verbindungen



Paketvermittlung — Datagramme

Pakete (= Datagramme) werden als isolierte Einheiten betrachtet
Ziel: In jedem Datagramm enthalten ⇝ keine Verbindungsverwaltung nötig
Routing: können Netz unterschiedlich durchlaufen
⇝ Datagramme können bei Empfänger unsortiert ankommen

Paketvermittlung — virtuelle Verbindungen

fester Übertragungsweg zwischen zwei Endsystemen
Reihenfolgetreue: Alle Pakete gehen selben Weg
Kennungen: *virtual circuit identifier* (VCI) kennzeichnen Pakete
Ziel: Zieladresse nur bei Verbindungsaufbau nötig
Prozess: Verbindungsaufbau → Datenübertragung → Verbindungsabbau

Vermittlungstechniken — Nachrichtenvermittlung

Vermittlung von Anwendungsnachrichten
Vermittlung üblicherweise mittels mehrerer Pakete
⇝ *Segmentierung* und *Reassemblierung*
Zwischensysteme: Reassemblierung nötig, da alle Teile in selbe Richtung weitergeleitet werden müssen
⇝ *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

Vermittlungsschicht — Aufgaben

Weiterleitung (*forwarding*): Datenebene
- Pakete werden von Routereingang an geeigneten Ausgabe weitergeleitet

Wegewahl (*routing*): Kontrollebene
- ermittelt Weg, den Pakete zurücklegen (sollen)
- erfordert Routingalgorithmus + -protokoll

Vermittlungsschicht — Kontroll- und Datenebene

Kontrollebene:
- betrachtet gesamtes Netz
- bestimmt wie Datagramm über Router von Quelle zu Ziel geroutet wird
- *Konzepte*:
  · traditionelle Routingalgorithmen: in jedem Router implementiert
  · *software defined networking* (SDN): in logisch zentralen Servern implementiert

Datenebene:
- Funktionen lokal in Router
- Bestimmt wie Datagramm von Eingangs- zu Ausgangsport geleitet wird

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 von Start zu Ziel

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

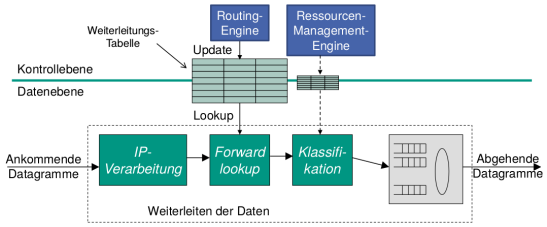
ARP (*address resolution protocol*):
Zuordnung von IP-Adressen zu Adressen der Sicherungsschicht

RARP *reverse ARP*:
Umkehrfunktionen von ARP

BGP (*border gateway protocol*), **RIP** (*routing information protocol*), **OSPF** (*open shortest path first*): Routingprotokolle

Vermittlung — IP

IP macht die ganze Vermittlung ⇝ nur ein großes Vermittlungsprotokoll
- Interoperabilität erhöhen
- Anzahl unterschiedlicher Interfaces erniedrigen
- kleinster gemeinsamer Nenner
- Anzahl nutzbarer Netze maximieren



IP — Fragementieren + Reassemblieren

Anpassung an Maximallänge unterliegender Netze (**MTU**: *maximum transfer unit*)

Flag-Felder IP-Kopf:
- *Bit 0*: reserviert, muss 0 sein
- *Bit 1*: 0 = darf fragmentiert werden, 1 = nicht
- *Bit 2*: 0 = letztes Fragment, 1 = nicht

IP — Weiterleitung

Endsystem:
- Rechner mit Zieladresse direkt verbunden → Datagramm direkt zustellen
- Sonst: Datagramm-Übergabe an Standardrouter

Router: Verwendung *Weiterleitungstabelle*
- *Zieladresse*
- *Next-Hop-Router*
- *Flags*, die Start und Ziel genauer klassifizieren
- *Netzschnittstelle*, auf die bei Endsystem das Datagramm ausgegeben werden soll

Weiterleitungstabelle: Identifikation des nächsten Systems auf Weg zum Ziel

IP — Empfangsprozess

Überprüfungen:
- *Kopflänge*
- *Datagrammlänge*
- *Versionsnummer* IP
- *Prüfsumme*
- *Lebenszeit*
- *Protokoll-Identifikation*
- *Adressklassen*

Fehlerfall: Benachrichtigung ICMP (*internet control message protocol*) — möglicherweise wird ICMP-Paket ausgesendet

IP — Adressierung

Ziel: Eindeutige Identifizierung aller angeschlossenen Systemschnittstellen
IP-Adressen: Kennungen für Interfaces von Routern/Endsystemen
- *IPv4*: 32 Bit-Adressen
- *IPv6*: 128 Bit-Adressen

IP — Subnetze

Gliederung: IP-Adresse unterteilt in
- *Subnetz-Teil*: high order bits
- *Endsystem-Teil*: low order bits

Subnetz: Interfaces mit selbem Subnetz-Teil, können ohne Router kommunizieren

Adresszuteilung

Manuell: Durch Systemadministrator
Dynamisch: *dynamic host configuration protocol* (DHCP)
- DHCP-Server liefert IP auf Anfrage

Adressblockzuteilung

Provider: Erhält Block von **ICANN** (*internet corporation for assigned names/numbers*)
- ICANN allokiert Adressen
- ICANN verwaltet DNS
- ICANN weist Domainnamen zu

Internet Control Message Protocol (ICMP)

Einzelne Datagrammverluste: meldet IP nicht (unzuverlässiger Dienst)
Schwerwiegende Probleme (z.B. Verbindungsabbruch): Mitteilung an Kommunikationspartner via ICMP
$\Rightarrow$ ICMP tauscht Fehlernachrichten, Statusanfragen und Zustandsinformationen aus

ICMP — Statusanfragen

Echo + Antwort (*echo and echo reply*):
- Aktivitätsüberprüfung von Kommunikationssystemen
- Empfänger von 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 von IPv4 geht aus
- Kopfformat IPv4 nicht optimal

Lösung:
- Erhöhung Adresslänge von 32 auf 128 Bit
- feste Kopflänge (40 Byte)
- keine Unterstützung von Fragmentierung
- keine Prüfsumme
- Optionen: als Erweiterungsköpfe (*next header*)
- ICMPv6: neue Version von ICMP

Routing — Prinzipien

Ziel: guten Weg durch Netz finden
Weg: Sequenz von Routern von Start- zu Ziel-Endsystem
Netzgraph: Netz wird als Graph verstanden
- *Knoten*: Router
- *Kanten*: Übertragungsabschnitte (Kantenkosten z.B. Verzögerung, Preis,…)

Pfad: Knotenfolge (meist Pfad mit kürzester Länge gesucht)

Routing-Verfahren — Dynamik

Frage: Wie dynamisch ist Routing-Verfahren?
Nicht adaptiv: Routen ändern sich sehr selten, wesentlich seltener als Verkehrsänderungen
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:
- Zufallszahl $1 < x \leq 0$
- Falls $x < 0.6$ weiterleiten nach B
- Falls $0.9 \geq x \geq 0.6$ weiterleiten nach C
- Sonst 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 $\rightsquigarrow$ 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 Systeme nah an RCC Tabellen schneller erhalten
- starke Belastung des RCC

Routing-Verfahren — isoliert

Prinzip: Jedes System entscheidet anhand selbstgesammelter Information
- kein Routinginformationsaustausch zwischen Systemen

Isoliertes Routing — Fluten

Prinzip: Jedes eingehende Datagramm wird auf jeder Übertragungsleitung weitergeleitet
Fluteindämmung:
- *Sequenznummern* für Duplikaterkennung
- *Lebensdauerkontrolle* durch Zählen der Übertragungsabschnitte (*hops*)

Varianten:
- *selektives Fluten*: Weiterleitung nicht auf allen Übertragungsabschnitten
- *random walk*: Zufällige Auswahl eines Übertragungsabschnittes

Isoliertes Routing — Hot Potato

Prinzip: Jeder versucht Datagramme so schnell wie möglich weiterzuleiten
$\rightsquigarrow$ 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
- *Problem*: kürzerer langsamer Weg wird längerem schnelleren vorgezogen

Link-State-Algorithmen: Unterschiedliche Routing-Metriken
- berücksichtigt aktuelle Zustände der Netzanschlüsse
- jeder Router kennt ganze Netztopologie
- Link-State- konvergieren schneller als Distanz-Vektor-Algorithmen

Routing-Algorithmen — Distanz-Vektor

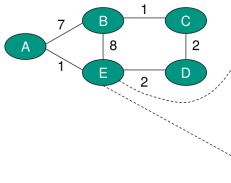
Eigenschaften:
- *verteilt*: jeder Router erhält Infos von direkten Nachbarn, führt Berechnung durch und verteilt dann neue Informationen an direkte Nachbarn
- *iterativ*: Verteilen + Berechnen von Informationen so lange, bis keine Information mehr ausgetauscht wird

Distanz-Vektor-Tabelle:
- Grundlegende, in jedem System vorhandene Datenstruktur
- Zeile für jedes mögliche Ziel
- Spalte für direkte Nachbarn

Distanz-Vektor-Routing — Distanz-Vektor-Tabelle

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 $D \to A\,(=3)$
  $\rightsquigarrow$ **minimale Kosten von D nach A über Nachbar von D**



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 (Bellman-Ford)

Initialisierung:
- für alle Nachbarn v: $D^X(*,v) = \infty$, $D^X(v,v) = c(X,v)$
- für alle Ziele y: sende $\min_w D^w(y,w)$ allen Nachbarn (w enthält alle 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 hat sich um α geändert
  $\rightsquigarrow D^X(Y,V) := c(X,V) + \alpha$
  $\rightsquigarrow$ Falls neuer $\min_w D^w(Y,w)$ für Ziel Y existiert, dann sende allen direkten Nachbarn diesen Wert

Komplexität: $O(n^2k)$ für n Knoten und k Kanten

Distanz-Vektor-Algorithmus — Updateausbreitung

Ausbreitung good news: schnelle Ausbreitung
Ausbreitung bad news: langsame Ausbreitung, ggf Routing-Schleifen
$\rightsquigarrow$ **Count to Infinity-Problem**

Distanz-Vektor-Algorithmus — poisoned reverse

Ziel: Vermeidung von Routing-Schleifen
Prinzip: Routing-Information wird Y vorenthalten, wenn Weg über Y kürzer

Routing-Algorithmen — Link-State-Routing

Prinzip:
- 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

Link-State vs. Distanz-Vektor

Komplexität Kontroll-Pakete:
- *link-state*: jedes System muss Kosten aller Links kennen, Änderungen müssen an alle Systeme geschickt werden
- *Distanz-Vektor*: Änderungen nur an benachbarte Systeme weitergegeben

Konvergenzgeschwindigkeit:
- *link-state*: $O(n^2)$, $O(nE)$ Pakete $\rightsquigarrow$ schnelle Konvergenz, danach schleifenfrei, aber Oszillationen möglich
- *Distanz-Vektor*: langsame Konvergenz, Schleifen möglich, Count-to-Infinity kann auftreten

Robustheit:
- *link-state*: Routenberechnungen separiert $\rightsquigarrow$ Robustheit
- *Distanz-Vektor*: ein System kann inkorrekte Pfade zu allen Zielen verbreiten

Gewinner?
- *link-state*: Konvergiert schneller, ist robuster
- *Distanz-Vektor*: einfacher zu implementieren

Software Defined Networking

Eigenschaften:
- **E1**: Separierung von Kontroll- und Datenebene
- **E2**: Flow-basierte Paketweiterleitung
- **E3**: Log an externen Controller ausgelagert
- **E3**: Netzwerk programmierbar

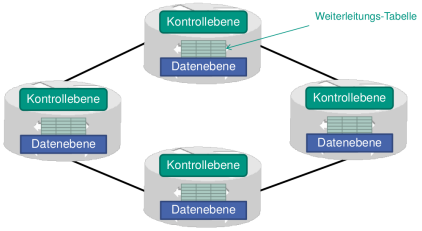
Umsetzung: *open flow*-Protokoll
- regelt Kommunikation zwischen Controller und Switch
- OpenFlow quasi-Standard, Alternativen existieren

Traditionelles IP-Routing

Kontroll- und Datenebene in jedem Router

Vorteile:
- Ausfallsicherheit (selbstorganisiert, verteilte Kontrolle, hohe Redundanz)
- Schnelle Reaktion (optimierte Routing-Hardware, lokale Routingentscheidung)
- Bewährtes Konzept

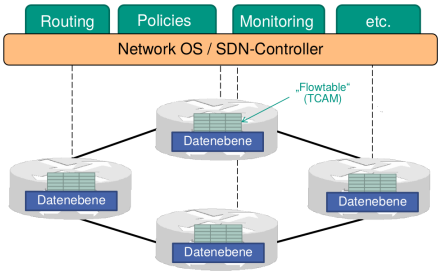
Nachteile:
- proprietäre Management-Schnittstellen (Mischbetrieb schwierig)
- unflexibel (neue Funktionen hinzufügen schwierig, aufwändige Standardisierung)
- teuer (hochqualifiziertes Personal + Overprovisioning benötigt)



SDN-Routing

Vorteile:
- logisch zentralisierte Sicht (Controller hat Netzüberblick, einfacher Einsatz von Graphalgorithmen)
- neue Funktionalität in Software (als App im Controller
  $\rightsquigarrow$ kürzere Entwicklungszyklen)
- Trennung von Kontroll- und Datenebene (Innovationen unabhängig möglich, herstellerunabhängig durch offene Schnittstellen)

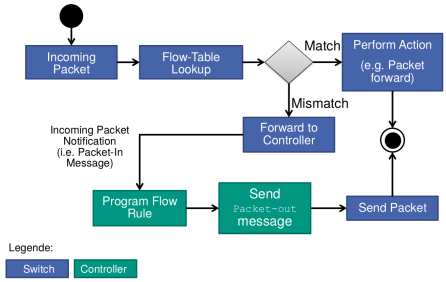
Nachteile:
- Skalierbarkeit
- *single point of failure*
- Kommunikation mit Controller langsam



SDN — Traditioneller Router vs. SDN-Switch

Traditioneller IP-Router:
- kennt keine Flows
- Weiterleitung über Ziel-IP-Adresse

SDN-Switch:
- Weiterleitung über Flowtable
- nutzt verschiedene IP-Kopf-Felder
- speichert Zustand pro Flow



V. Sicherungsschicht

Terminologie

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

Aufgaben

Strukturierung des Datenstroms (*framing*)
$\rightsquigarrow$ Datagramm in Rahmen einkapseln
Medienzugangskontrolle bei geteilten Medien
Adressierung mittels MAC-Adressen
Je nach angebotenem Dienst *Fehlererkennung/-behebung* bzw. *Flusskontrolle*
Übertragungsart:
- (Semi-) Broadbcast
- Punkt-zu Punkt (Halb-/Vollduplex)

Broadcast- vs. Punkt-zu-Punkt-Link

Broadcast-Link:
alle Stationen können alle gesendeten Rahmen sehen (zB WLAN = semi-broadcast)
Punkt-zu-Punkt-Link:
zwei Stationen sind über dedizierten Link verbunden (zB switch-basiertes Ethernet)

Punkt-zu-Punkt-Kommunikation

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 — Kommunikation

Sender:
- Datagramm einkapseln
- ggf Felder für Prüfsumme, Flusskontrolle etc hinzufügen

Empfänger:
- Überprüfung hinsichtlich Fehler, Fluskontrolle usw
- Datagramm extrahieren und an Vermittlungsschicht weiterreichen

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 senderseitig Sicherungssequenz zur Überprüfung auf Empfangsseite
- *frame check sequence* (FCS)
- steht üblicherweise an Rahmenende als Anhang

Fehlererkennung — cyclic redundancy check (CRC)

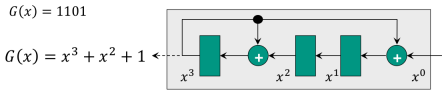
Polynom: $0101 \rightarrow 0x^3 + 1x^2 + 0x^1 + 1x^0 = x^2 + 1$
Generatorpolynom: von $g(x)$ generierte Code ist
$C := \{v(x) \mid \deg(v(x)) < n \wedge g(x) \text{ teilt } v(x)\}$
Prinzip:
- gleiches Polynom $G(x)$ für Sender und Empfänger
- *Sender*:
  · hängt $\deg(G(x))$ Nullen an Daten
  · berechnet Rest von $M(x)/G(x)$ (m Bit Rahmen $\rightarrow M(x)$)
  · hängt Rest an mit Nullen erweiterte Daten an
- *Empfänger*: Dividiert durch $G(x)$
  · Ergebnis 0: keine Fehler erkannt
  · Ergebnis $\neq 0$: Fehler!

CRC — Wichtige Generatoren

CRC-12: $x^{12} + x^{11} + x^3 + x^2 + x + 1$
CRC-16: $x^{16} + x^{15} + x^2 + 1$
CRC-CCITT: $x^{16} + x^{12} + x^5 + 1$
CRC-32: $x^{32} + x^{26} + x^{23} + x^{22} + x^{16} + x^{12} + x^{11} + x^{10} + x^8 + x^7 + x^5 + x^4 + x^2 + x + 1$

CRC — Hardwareimplementierung

Rückgekoppelte Schieberegister $\rightarrow$ CRC bei Durchschieben berechnet
Prinzip:
- Bitweises Empfangen der Daten, durchlaufen Schieberegister
- Rückkopplung durch XOR-Gatter an 1-Stellen des Generators (ohne höchstes Bit)



Multiplexing

Problem: Link von mehreren Knoten parallel benutzt
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

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
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

Medienzugriff

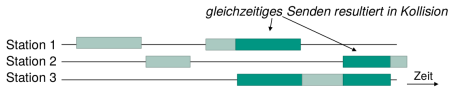
Problem: Unterschiedliche Medien (Kabel + Drahtlos)
Varianten:
- feste Mediumszuteilung (feste Zeitschlitze, Punkt-zu-Punkt-Verbindungen)
- konkurrierende Nutzung ⟶ Zugriffsorganisation notwendig

Zeitmultiplex — Kategorien

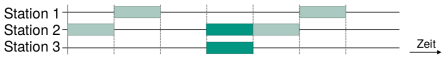
fest
variabel:
- *kontrollierter* Zugriff
  · zentral
  · dezentral
- *zufälliger* Zugriff

Zeitmultiplex — Zufallsstrategien

Aloha:
- verwendbar bei zufälligen, unabhängigen, seltenen Sendewünschen
- 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
- *Problem*: mehrere Systeme können quasi gleichzeitig Senden beginnen ⇝ Kollisionen

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*)

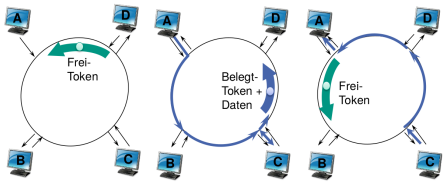
Kollisionsfreier Zugriff — Prinzip

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

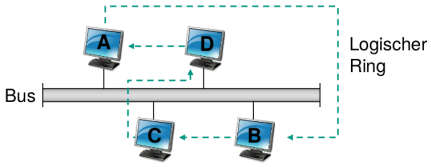
Prinzip:
- Systeme physikalisch Punkt-zu-Punkt-verbunden zu Ring
- Jedes System hat *Vorgänger* und *Nachfolger*
- Senderechtszuteilung durch zirkulierendes Token



Kollisionsfreier Zugriff — Token Bus

Prinzip:
- Verbindet Vorteile von Ethernet und Token Ring
- Busverkabelung wie bei Ethernet
- *Garantierte Antwortzeiten* durch zirkulierendes Token

Aufbau:
- Alle Stationen physikalisch durch Bus verbunden
- Bildung eines *logischen Rings*



Lokale Netze — MAC-Adressen

Theoretisch weltweit eindeutig
Aufbau:
- 24 Bit von IEEE an Hersteller zugewiesen
- 24 Bit von Hersteller durchnummeriert

Funktion: lokal genutzt, um Rahmen von Interface zu benachbartem, physikalisch verbundenem Interface zu übertragen

Format:
- 48 Bit
- 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?
Aufgabe: MAC-Adresse zu bekannter IP-Adresse ermitteln
Prinzip: dynamisch Adresszuordnungen lernen
ARP-Cache: kleine Tabelle auf jedem System
- Eintrag IP + MAC + maximale Lebenszeit
- Einträge bei Bedarf gelernt

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 — falls eigene IP ARP-Reply
  · 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 und Sender-MAC auf R
4. Router leitet Datagramm weiter

Lokale Netze — Ethernet

Standard: IEEE 802.3
Medienzuteilung:
- zeitmultiplex, variabel, zufälliger Zugriff
- Verwendung von CSMA/CD (exponentieller Backoff)

Netztopologie: Ursprünglich Bus-, heute Sterntopologie
Varianten:
- *Bezeichnung*: [Datenrate][Baseband/Broadband][Medium]
- *Invadiante*: Format Ethernet-Rahmen
- *10Base5*: 10Mbit/s, Baseband, Bustopologie, 10mm Koax
- *10Base2*: 10Mbit/s, Baseband, Bustopologie, 5mm Koax

Ethernet

	10Base5	10Base2	10Base-T
Medium	Koaxialkabel		Twisted Pair
Kodierung	Manchester		
Topologie	Bus		Stern

Fast Ethernet

	100Base-T	100Base-T4	100Base-Tx	100Base-Fx
Medium	Twisted Pair			Glasfaser
Kodierung	Manchester	8B/6T NRZ	4b/5B NRZI & MLT-3	4B/5B NRZI
Topologie	Stern			

Gigabit Ethernet and beyond

	1000Base-SX	1000Base-T	10GBase-SR	10GBase-T
Medium	Glasfaser	Twisted Pair	Glasfaser	Twisted Pair
Kodierung	8B/10B NRZ	PAM-5 & Trellis	66B/68B	PAM-16 & DSQ128
Topologie	Stern			

Ethernet — exponentieller Backoff

Schema: Station wählt *randomisiert* Anzahl zu wartender Zeitschlitze nach Schema:
- *1. Kollision*: Wartezeit 0/1 Zeitschlitze
- *2. Kollision*: Wartezeit 0/1/2/3 Zeitschlitze
- *i. Kollision*: Wartezeit 0/…/$2^i - 1$ Zeitschlitze
- $i = 16 \rightsquigarrow$ Systemfehler

Ethernet — Zeitschlitze

Prinzip:
- Kanal wird logisch in Zeitschlitze fester Länge aufgeteilt
- Dauer = minimale Rahmenlänge $\rightarrow$ Kollisionserkennung vor Zeitschlitz-Ende

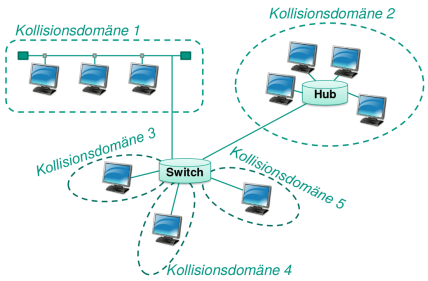
Ethernet — Switches

Prinzip:
- Schicht-2-Netzkopplung
- Trennung von Inter- und Intranetz-Verkeht $\rightarrow$ Erhöhung Netzkapazität
- Switches nicht sichtbar für Endsysteme

Ziel: Selbstorganisierte Netzkonfiguration mit Switches
Aufgaben:
- Etablierung schleifenfreie Netztopologie (*spanning tree*)
- Etablierung von Wegen zwischen Endsystemen (selbstlernend)

Ethernet — Kollisionsdomänen

= Netzbereich, auf dem Kollision möglich ist



Virtual Local Area Network (VLAN)

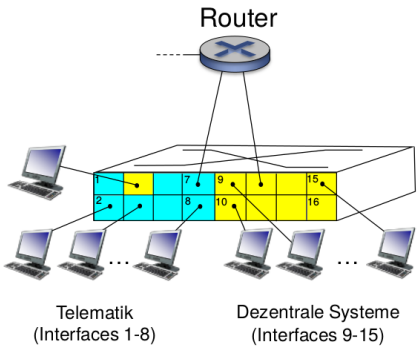
Idee: Logische Trennung von Datenverkehr auf Ethernet-Ebene
$\rightsquigarrow$ virtuelle Leitung
Sicherheit:
- Trennung in logische Medien ermöglicht gezielte Systemgruppierung
- Bessere Kontrolle über Netzstruktur

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

Verkehrsisolation: Rahmen von Interfaces 1-8 können nur Interfaces 1-8 erreichen
$\rightsquigarrow$ Sicherheit, Performance
Dynamische Zuweisung: Interfaces dynamisch anderen VLANs zuordnern
$\rightsquigarrow$ Flexibilität
Weiterleitung zwischen VLANs über Routing (oft über in Switch integrierten Router)
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



VI. Architektur

OSI-Referenzmodell

Schicht 1: Bit-Übertragungsschicht (*physical layer*)
- Bitübertragung
- Verwendung von Leitungscodes usw
- *keine* Pufferung, *kein* zuverlässiger Dienst
- *Ziel*: Übertragungsqualität

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

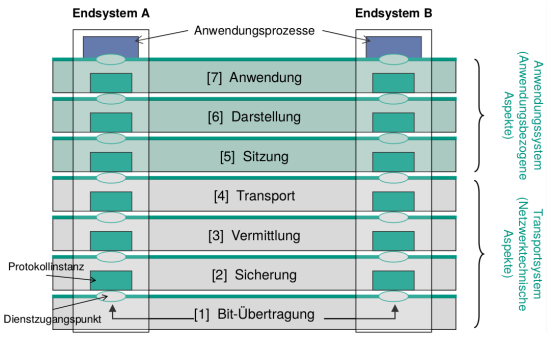
Schicht 3: Vermittlungsschicht (*network layer*)
- *Ziel*: Verknüpfung von Übertragungsabschnitten zu Netz
- Wegewahl im Kommunikationssystem
- Geräteadressierung
- Multiplexing

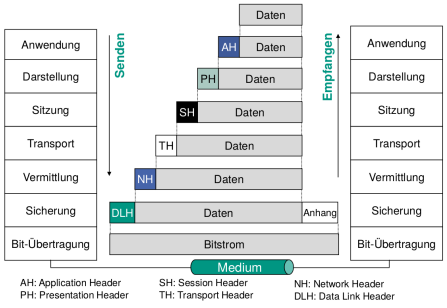
Schicht 4: Transportschicht (*transport layer*)
- *Ziel*: Übertragung von Daten zwischen Anwendungen
- Abstrahiert von Diensten der Vermittlungsschicht
- Fehlererkennung/-behebung
- Pufferung
- Adressierung von Transportdienstnutzern
- Multiplexing

Schicht 5: Sitzungsschicht (*session layer*)
- *Ziel*: Nichtunterbrechbarkeit von Kommunikationsbeziehungen
- Datenauschtauschsgliederung nach Gesichtspunkten der Anwendung

Schicht 6: Darstellungsschicht (*presentation layer*)
- *Ziel*: Einheitliche Datendarstellung
- Kommunikation zwischen heterogenen Geräten
- Beibehaltung der Informationssemantik

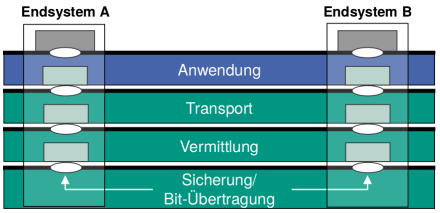
Schicht 7: Anwendungsschicht (*application layer*)
- *Ziel*: Austausch anwendungsabhängiger Daten

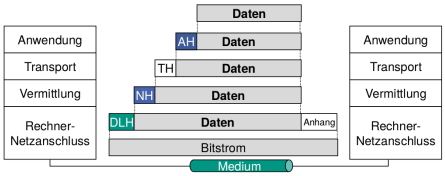




Internet-Referenzmodell

Einfachereres Modell, nur 4 Schichten (manchmal 5 bei Trennung von Sicherung und Bit-Übertragung)
Schichten gehen über gesamtes Kommunikationssystem, nicht nur auf einem Gerät





Protokolle und Dienste

Protokoll: Regeln und Formate für Datenaustausch *innerhalb* einer Schicht
Dienst:
- Zusammenwirkung von Protokollinstanzen *innerhalb* einer Schicht
- Bündelung zusammengehöriger Funktionen
- *Dienstfunktion*: einzelne Dienstteile können 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

Horizontale und vertikale Kommunikation

Horizontale Kommunikation:
- zwischen Sender und Empfänger
- Protokollinstanzen einer Schicht tauschen Daten aus um Dienst zu erbringen

Vertikale Kommunikation
- zwischen Schichten
- Protokollinstanz Schicht n greift auf Dienste von Protokollinstanz Schicht n-1 zu
- Protokollinstanz Schicht n gibt Daten an Protokollinstanz Schicht n+1 weiter

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

VII. Sicherheit

Übersicht

Bausteine: Signatur, Verschlüsselung, Hashfunktion, Message Authentication Code, Schlüsselaustausch,...
Schutzziele: Vertraulichkeit, Integrität, Authentizität, Privatheit, (Nicht)-Abstreitbarkeit,...
Angriffe: Abhören, Einfügen, Manipulieren, Man in the Middle, Replay, Denial of Service,...
System/Protokoll: TLS/SSL, IPsec, Kerberos,...
⇝ **System/Protokoll** verwendet **Bausteine** um **Schutzziele** zu realisieren und vor **Angriffen** zu schützen

Schutzziele

Anforderungen an eine Komponente oder ein System, die erfüllt werden müssen, um schützenswerte Güter vor Bedrohen zu schützen
Kategorien:
- **C**onfidentiality
- **I**ntegrity
- **A**vailability

weitere Schutzziele:
- Authentizität
- Privatheit
- (Nicht-)Abstreitbarkeit

Schutzziele — Vertraulichkeit

Ein System bewart Vertraulichkeit, wenn es keine unauthorisierte Informationsgewinnung ermöglicht
Bausteine: (A)-Symmetrische Verschlüsselung

Schutzziele — Integrität

starke Integrität: *Ein System bewahrt starke Integrität, wenn es nicht möglich ist, Daten unauthorisiert zu manipulieren*
schwache Integrität: *Ein System bewahrt schwache Integrität, wenn unauthorisierte Manipulationen an Daten nicht unbemerkt möglich sind*
Bausteine: Tamper Proof-Module, Message Authentication Codes (MAC)
Probleme:
- Manipulationen in vielen Fällen nicht zu verhindern
  ⇝ wenigstens nicht unbemerkt ⇝ **schwache Integrität**

Angriffe: Ersetzen, Einfügen oder Löschen von Daten

Schutzziele — Authentizität

Die angegebene Datenquelle ist tatsächliche Quelle + Datenintegrität
Subjektechtheit: Bob will sicherstellen, dass er tatsächlich mit Alice kommuniziert
⇝ **Authentifikation**
Datenechtheit: Bob will sicherstellen, dasss Daten tatsächlich von Alice sind
Bausteine: Zertifikate, Signaturen, gemeinsames Geheimnis

Schutzziele — Verfügbarkeit

Beschreibt, in welchem Maße die Systemfunktionalität von berechtigen Subjekten unabhängig von Einflüssen in Anspruch genommen werden kann

Angriffe

Sniffing, Denial of Service,…

Bausteine

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

Integritätssicherung:
- **kryptografische Hashfunktion**: verwendet symmetrische Kryptografie
- **digitale Signatur**: verwendet asymmetrische Kryptografie

Schlüsselaustausch
Message Authentication Code:
- Ziel: Empfänger erkennt Manipulation an empfangenen Daten
- Voraussetzung: Alice und Bob haben gleichen symmetrischen Schlüssel
- Vorgehensweise: Alice hängt Hash von Nachricht + Schlüssel an Nachricht an

Digitale Signatur:
- Ziel: Bob kann verifizieren, dass nur Alice nur dieses Dokument unterschrieben hat
- Vorgehensweise: Hash des Dokuments mit privatem Schlüssel verschlüsseln, als Signatur mitsenden, entschlüsseln mit öffentlichem Schlüssel

Digitales Zertifikat:
- Ziel: Verifizieren, dass jemand der ist, für den er sich ausgibt
- Problem: kann man nicht selbst überprüfen ⇝ verlassen auf Dritte
- Vorgehensweise: Überprüfung durch *certificate authority* (CA), Zertifikat = Authentifikation des öffentlichen Schlüssels

Sicherheitsprotokolle