Einführung

Was ist das Internet?

Komponentensicht
1. *Computer* – führen Netzwerkanwendungen aus
2. *Kommunikationsmedien* – Kupferkabel, Glasfaser, Funk
3. *Zwischensysteme* – Weiterleitung durch Router und Switches

Dienstsicht
⇒ Infrastruktur, die Dienste bereitstellt
- *Kommunikation* (Mail, Messaging, soziale Medien)
- *Information* (Surfen)
- *Unterhaltung* (Streaming, Spiele)

Rand des Internet

Geräte
- Clients
- Server

Zugangsnetze
- Heimnetz
- Mobiles Zugangsnetz
- Unternehmensnetz

Kern des Internet

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

Anwendungsschicht

Historie

70er/80er:
- textbasierte Anwendungen

90er:
- World Wide Web
- Instant Messaging
- P2P-Filesharing

seit 2000: steigende Vielfalt + Allgegenwärtigkeit
- Streaming (Spotify, YouTube)
- Gaming
- Soziale Netzwerke
- Smartphones

Grundlagen — Schichtenmodell

Kommunikation in Schichten organisiert
Anwendungsschicht: oberste Schicht
- enthält Anwendungsprotokolle
- Anwendung kümmert sich nicht um Datentransport

Datentransport: unter Anwendungsschicht liegende Schichten
- Interna für Anwendung transparent
- Verzögerungen bleiben vor Anwendung verborgen

Grundlagen — Verzögerung

Abhängig von
- *Ausbreitungsverzögerung* t_a
- *Sendezeit* t_s
- *Pufferfüllstände*

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!
  ⇝ Ausbreitungsverzögerung t_a

Verzögerung im Router
- *Pufferung* der Daten in Warteschlange
- *Verarbeitung* (Fehlerüberprüfung usw.)

Grundlagen — Protokollstack

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

Grundlagen — Prozess und Nachricht

Prozess: Programm, das im Endsystem (Anwendungsschicht) abläuft
Nachricht: Ausgetauscht zwischen Prozessen auf *unterschiedlichen* Endsystemen

Grundlagen — 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 Endsystem kommunizieren gleichzeitig über Netzwerk
  ⇝ eindeutige Socket-Identifikation über Portnummer

Grundlagen — Client-Server-Anwendungen

Server:
- ständig in Betrieb
- permanente IP-Adresse
- häufig in Datenzentren

Clients:
- kommunizieren mit Server
- kommunizieren *nicht* direkt miteinander
- evtl. nicht immer verbunden
- evtl. dynamische IP-Adresse

Grundlagen — Peer-to-Peer-Anwendungen

Endsysteme kommunizieren direkt miteinander
- fordern Dienste von anderen Peers an
- nicht permanent verbunden, wechseln dymanisch IP-Adressen
  ⇝ komplexes Management

selbst-skalierend
- neue Peers erhöhen Kapazität, fordern aber auch selber 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 — Überblick

Protokoll der Anwendungsschicht (*hypertext transfer protocol*)
- einfaches, ASCII-basiertes Transferprotokoll

Basiert auf Client/Server-Modell
- *Client*: Browser, der Web-Objekte anfordert, empfängt und darstellt
- *Server*: sendet über HTTP angeforderte Objekte

zwei Nachrichten-Typen: *Request*, *Response*
Zustandslos:
- jeder Request wird individuell bearbeitet
- keine Zustandsinformation auf dem Server

nutzt TCP zur Kommunikation
1. Client initiiert Verbindungsaufbau
2. Server akzeptiert Verbindung
3. Austausch von HTTP-Nachrichten
4. Abbau der TCP-Verbindung

HTTP — Methoden

HTTP-Anfragen können verschiedene Methoden nutzen
GET: Resource von Server zu Client übertragen (z.B. normale Webseite)
POST: Daten zu Ressource übertragen (z.B. Web-Formular)
Weitere Methoden:
- PUT — neue Ressource anlegen
- DELETE — Ressource löschen
- HEAD — wie GET, aber nur HTTP-Header übertragen
- …

HTTP — 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

Non-persistent HTTP:
- höchstens ein Objekt wird über TCP-Verbindung gesendet, danach geschlossen
  ⇝ Herunterladen mehrerer Objekte erfodert mehrere TCP-Verbindungen

Persistent HTTP: mehrere Objekte über eine TCP-Verbindung

Non-persistent HTTP — Antwortzeit

Round Trip Time (RTT): Zeit, die Paket von Sender zu Empfänger und zurück benötigt
HTTP-Antwortzeit:
- ein RTT für Verbindungsaufbau
- ein RTT für HTTP-Anfrage und erste Antwortbytes
- Zeit t_s für Senden der Datei
  ⇝ **Antwortzeit** $2 * \text{RTT} + t_s$

Cookies

Speichert Nutzer-Server-Zustand
⇝ Server kann Inhalt abhängig von Nutzeridentifikation bereitstellen
Komponenten:
- Cookie-Information in HTTP-Response-Nachricht
- 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

Cookies — Privatsphäre

Webseiten unterscheiden Nutzer durch Cookies
  ⇝ Werbeanbieter können Nutzer über viele Webseiten tracken
Webseiten können durch Cookies sehr viel über Nutzer lernen

Mail — Komponenten

User Agent (UA):
- lesen, senden, weiterleiten
- Beispiele: Outlook, Thunderbird

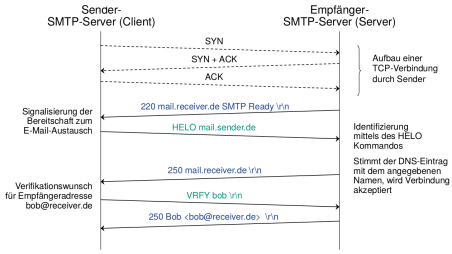
Mailserver:
- *mail transfer agent* (MTA)
- *mail delivery agent* (MDA)
- User-Mailboxen

simple mail transfer protocol (SMTP)
- Client/Server-Modell
- Transfer von Mails vom User Agent zum Mailserver

SMTP — Aufbau

Drei Phasen:
1. Handshake
2. Nachrichtenübermittlung
3. Abschluss

Command/Response-Interaktionen
- ähnlich Request/Response bei HTTP
- Kommandos: ASCII-Text
- Antwort: Statuscode + Nachricht



MIME

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

Mail — Postfach-Abfrage

POP3 (*post office protocol 3*):
- Client holt von Mailserver empfangene/gespeicherte Nachrichten ab
- einfache Funktionalität
- verwaltet Nachrichten im UA, *keine* Synchronisation zwischen mehreren UAs

IMAP (*interactive mail access protocol*):
- Nachtichten werden zentral auf Mailserver verwaltet
- erweiterte Kommandos (Ordner, Filter)

XMPP

Echtzeit-XML-Streaming-Protokoll
Grundlage für Whatsapp usw.
Dezentral, ähnlich wie E-Mail
Clients: zu ihrem jeweiligen Server verbunden
Server: verbinden sich untereinander zur Nachrichtenübermittlung
Adressformat:
- *Nutzer*: Server + Username, z.B. alice@jabber.org
- *Clients*: pro Nutzer, z.B. alice@jabber.org/laptop

DNS — Grundlagen

Ziel: Verwendung von Namen statt IPs
Aufgabe: Zuordnung IP-Adresse ↔ Name
Funktionalitäten:
- *Registrierung* von Namen + IP-Adressen
- *Auflösung* von Namen in IP-Adressen

DNS — Aufbau

Verteilte Datenbank von Name-Servern (DNS-Servern)
- Client-Server-Modell
- Server kann Anfrage an weitere Server weiterleiten

Protokoll der Anwendungsschicht
- Über Port 53 (UDP) realisiert

Basisdienst ⇝ keine Anwendung
- Komplexität am Rande des Netzes lokalisiert
  → Internet-Design-Philoshopie!

DNS — Anfragen

Rekursiv: kennt angefragter Server Antwort nicht, fragt dieser dahinterliegende Server, bis er Antwort bekommt
Iterativ: kennt angefragter Server Antwort nicht, fragt *Client* andere Server
Üblich: Client fragt lokalen Name-Server rekursiv, dieser dann iterativ

DNS — Resource Records (RR)

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

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

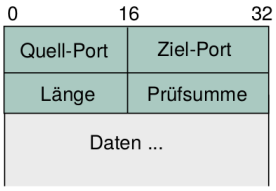
IP-Adressen

= *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

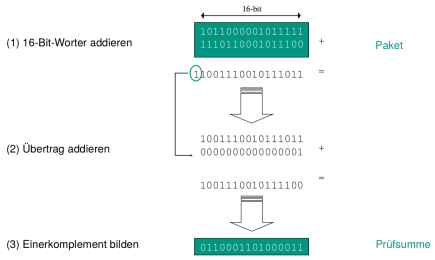
Wiederholungsaufforderung (*automatic repeat request*, ARQ)
- Empfänger teilt Sender Ergebnis der Fehlerkorrektur mit

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

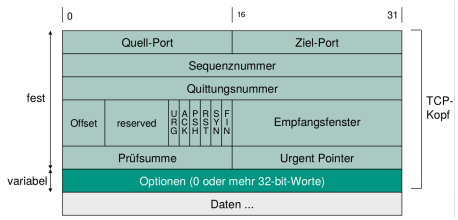
Flusskontrolle — Kreditbasiert

Prinzip:
- Sender kann höchstens n Pakete unquittiert senden
- n = Pufferkapazität des Senders ⇒ **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)

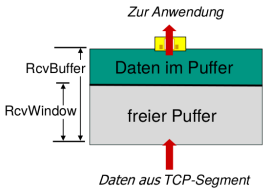


TCP — Felder

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

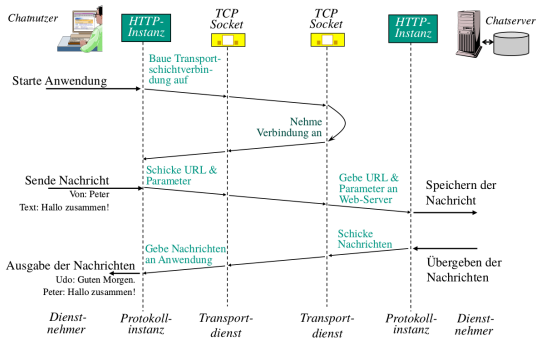
Ziel: Empfängerüberlastung vermeidne
Prinzip:
- *Empfänger*: reserviert Pufferplatz pro Verbindung (explizite Kreditvergabe)
  - `RcvBuffer`: gesamter Pufferplatz (default 4096 Byte)
  - `RcvWindow`: freier Pufferplatz (Empfangsfenster)
  - Sender sendet nicht mehr unbestätigt als in `RcvWindow` passt



Verbindungsverwaltung — 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:
  - LastByteSent − LastByteAcked $\leq$ min{CWnd, 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`