# 6. Die Transportschicht

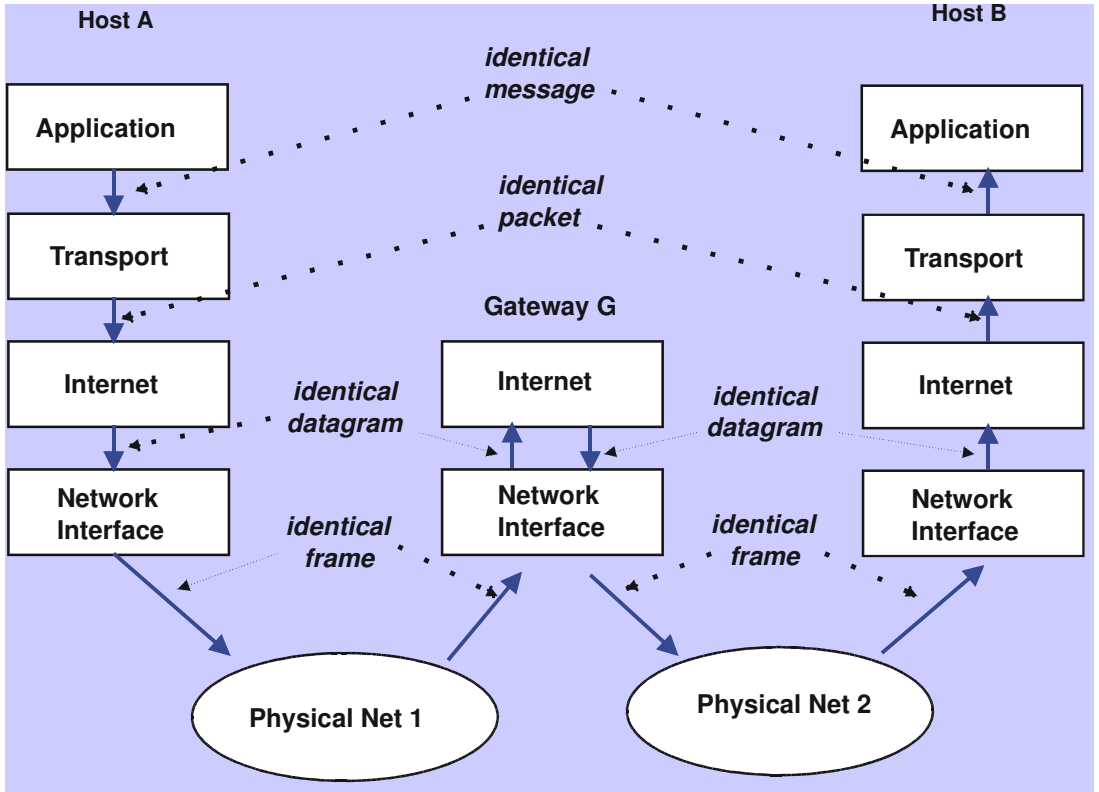

6.1 Architektur der Transportprotokolle im Internet

6.2 UDP (User Datagram Protocol)

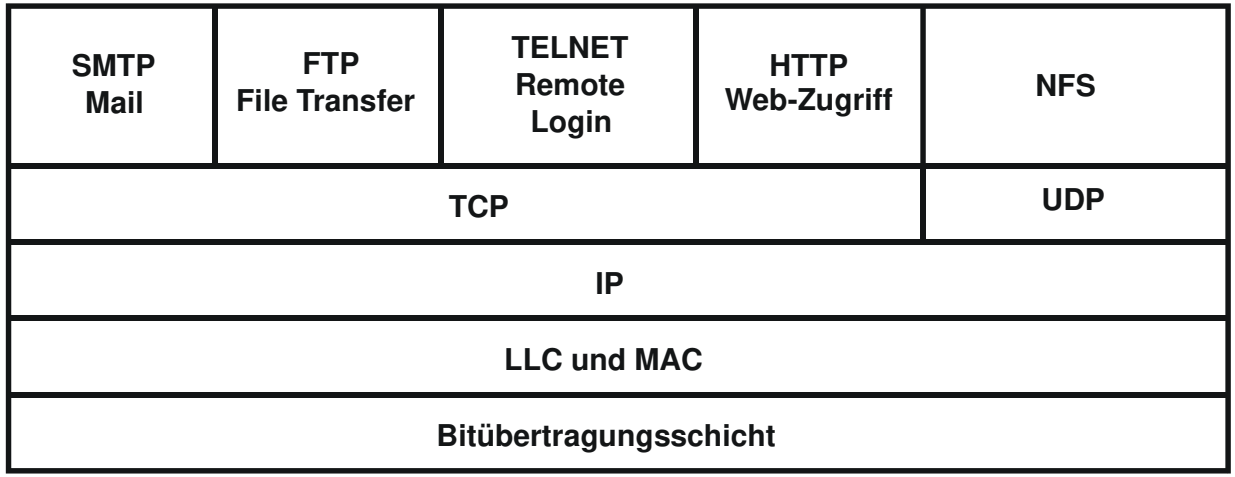
6.3 TCP (Transmission Control Protocol)

# 6.1 Architektur der Transportprotokolle im Internet

Die Transportprotokolle im Internet sind **Ende-zu-Ende-**Protokolle!



# Wichtige INTERNET-Protokolle




| | | |
|---|---|---|
| SMTP | = | Simple Mail Transfer Protocol |
| FTP | = | File Transfer Protocol |
| TELNET | = | Remote Login Protocol |
| UDP | = | User Datagram Protocol |
| NFS | = | Network File System |
| TCP | = | Transmission Protocol |
| IP | = | Internet Protocol |
| LLC | = | Logical Link Control |
| MAC | = | Media Access Control |

# Adressierung von Dienst-Prozessen: Ports

- Die Schicht-4-Adresse soll einen bestimmten Dienst (Anwendungstyp) identifizieren, dem ein Anwendungsprozess zugeordnet ist
- Eine Adressierung mittels Prozess-Nummer wäre ungeeignet, denn Prozesse werden dynamisch erzeugt und beendet; deshalb ist die Prozess-Nummer außerhalb des lokalen Systems nicht bekannt.
- Die Zuordnung Dienst <=> Prozess ist nicht fest:
  - Ein Prozess kann mehrere Dienste erbringen.
  - Mehrere Prozesse können denselben Dienst erbringen.

=> Einführung eines abstrakten Kommunikations-Endpunktes: **Port**

# Eigenschaften der Ports

- Ein Dienst ist genau einem Port zugeordnet.
- Über denselben Port können mehrere Verbindungen gleichzeitig laufen.
- Asynchroner und synchroner Port-Zugriff sind möglich.
- Jeder Port ist mit einem Puffer assoziiert.
- Der Port stellt eine Programmierschnittstelle für den Anwendungs-programmierer zur Verfügung (API).

# Beispiele für reservierte Port-Nummern

## Einige Port-Zuordnungen ("well-known addresses")

| Dezimal | Schlüssel-wort | Unix Schlüssel-wort | Beschreibung |
|---|---|---|---|
| 0 | | | Reserved |
| . | . | . | . |
| . | . | . | . |
| 20 | FTP-DATA | ftp-data | File Transfer Protocol (data) |
| 21 | FTP | ftp | File Transfer Protocol |
| 23 | TELNET | telnet | Terminal Con-nection |
| 25 | SMTP | smtp | Simple Mail Transfer Proto-col |
| 42 | NAME-SERVER | name | Host Name Server |
| 43 | NICNAME | whois | Who Is |

# 6.2 UDP (User Datagram Protocol)

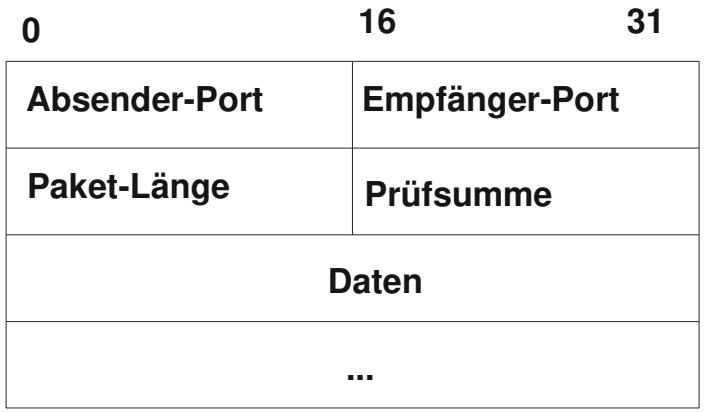
**UDP** ist ein **unzuverlässiges, verbindungsloses** Datagramm-Transport-Protokoll im Internet. Es dient im Wesentlichen dazu, den Anwendungen eine Programmierschnittstelle für den Direktzugriff auf IP, ergänzt um die Port-Adressierung, anzubieten.

## Eigenschaften

- Datagramm-Transport
  - Keine Garantie über das Einhalten der Reihenfolge der einzelnen Pakete
  - Keine gesicherte Zustellung der Pakete an den Empfänger
  - Duplizierte Pakete sind möglich
- Multicast ist möglich

# Format von UDP-Paketen

## UDP-Header




- Die Paketlänge wird in Bytes angegeben (einschließlich UDP-Header)
- Prüfsumme über Header und Daten zur Fehlererkennung. Die Prüfsumme wird über das gesamte Fragment berechnet, inklusive UDP-Header. Es wird ein spaltenweises EXOR über die „senkrecht untereinander geschriebenen“ 16-Bit-Worte des Pakets berechnet.
- Die Verwendung der Prüfsumme ist optional (wenn IPv4 darunter liegt)

# Eigenschaften von UDP

- Geringer Ressourcen-Verbrauch (Speicherplatz, CPU-Zeit)
- Kein expliziter Verbindungsaufbau
- Einfache Implementierung

UDP ist vor allem für einfache Client-Server-Interaktionen geeignet, zum Beispiel für Request-Response-Protokolle:

- ein Anfrage-Paket vom Client zum Server
- ein Antwort-Paket vom Server zum Client

# Anwendungsbeispiele für UDP

- Domain Name Service (DNS)
- SNMP: Simple Network Management Protocol
- NFS: Network File System
- viele Multimedia-Protokolle, die keine Fehlersicherung in Schicht 4 wollen
- alle Multicast-Protokolle, insbesondere auch RTP für Realzeit-Anwendungen, vor allem Multimedia-Anwendungen (Audio- und Videoströme)

# 6.3 TCP (Transmission Control Protocol)

TCP ist das erste Protokoll in der Internet-Protokollhierarchie, das eine **gesicherte Datenübertragung zwischen Endsystemen** leistet.

## Eigenschaften

- Datenstrom-orientiert: TCP überträgt einen seriellen Bit-Strom der Anwendung in Form von 8-Bit Bytes.
- **Verbindungsorientiert**: Vor der Datenübertragung wird eine Verbindung zwischen beiden Kommunikationspartnern aufgebaut, die fehlergesichert und Reihenfolge erhaltend ist.
- Gepufferte Datenübertragung: Der sequenzielle Datenstrom wird zur Übertragung in einzelne Segmente (Pakete) aufgeteilt.
- Duplex-Kommunikation: Über eine TCP-Verbindung können gleichzeitig Daten in beide Richtungen übertragen werden.

# Was steht im TCP-Standard?

Der TCP-Standard (RFC 793) spezifiziert

- die Formate von Datenpaketen und Kontrollinformationen
- die Prozeduren für
  - Verbindungsaufbau und -abbau
  - Fehlererkennung und -behebung
  - Flusskontrolle
  - Netzwerk-Überlastkontrolle (Congestion Control)

RFC 793 spezifiziert **nicht** die Schnittstelle zum Anwendungsprogramm (*socket*).

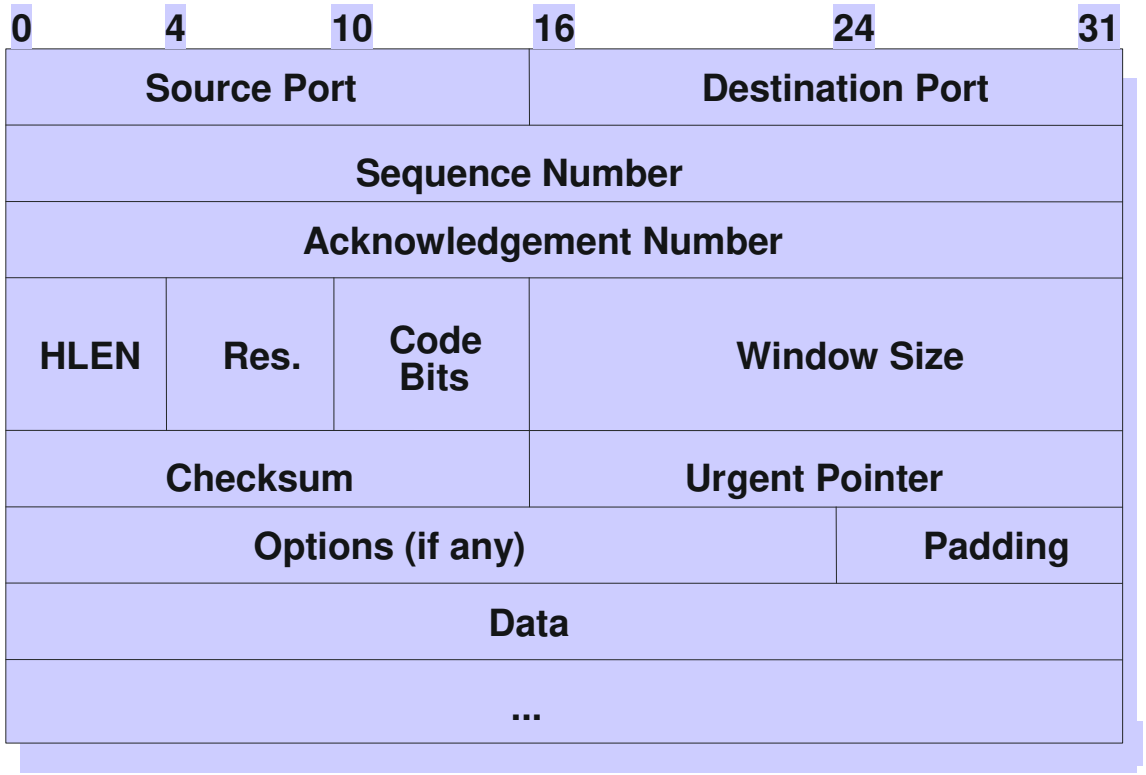
# Adressierung

Eine TCP-Verbindung ist eindeutig bestimmt durch ein Quintupel aus

- IP-Adressen von Sender und Empfänger
- Port-Adressen von Sender und Empfänger
- TCP-Protokoll-Identifikator

# Paketformat

## Format des TCP-Headers



# Datenfelder im TCP-Header (1)

| | |
|---|---|
| **SEQUENCE NUMBER**<br>**ACKNOWLEDGMENT** | Bytenummern |
| **HLEN** | headerlänge = Offset des Datenfeldes |
| **CODEBITS** | (6 Bits von links nach rechts) |
| URG | Urgent Pointer wird verwendet |
| ACK | Ack-Nummernfeld ist gültig |
| PSH | Push |
| RST | Reset der Verbindung |
| SYN | Synchronisiere Sequenznummern |
| FIN | Ende des Datenstroms |
| **WINDOW SIZE** | Fenstergröße in Bytes |
| **URGENT POINTER** | Byte-Offset zur aktuellen Sequenz-nummer, an der wichtige Daten beginnen |

# Datenfelder im TCP-Header (2)

- **Port-Nummern:** wie für UDP
- **sequence number:** identifiziert das erste Byte im Datenteil des Paketes
- **acknowledgement number:** identifiziert das nächste Byte im Datenstrom, das der Sender dieses Paketes vom Empfänger dieses Paketes erwartet. Nicht einzelne Datenpakete, sondern Byte-Positionen im Datenstrom werden bestätigt. Dadurch ist Kumulation von Bestätigungen über mehrere TCP-Pakete leicht möglich. Die acknowledgements steuern die Übertragungswiederholung im Fehlerfall. Sie definieren auch das Schiebefenster für die Flusskontrolle. Da Daten in beide Richtungen zwischen den Kommunikationspartnern fließen können (Duplex-Betrieb), gibt es eine eigene Sequenznummernfolge für jede Richtung.
- **header length (HLEN):** Größe des TCP-Headers in 32-Bit-Worten

# Datenfelder im TCP-Header (3)

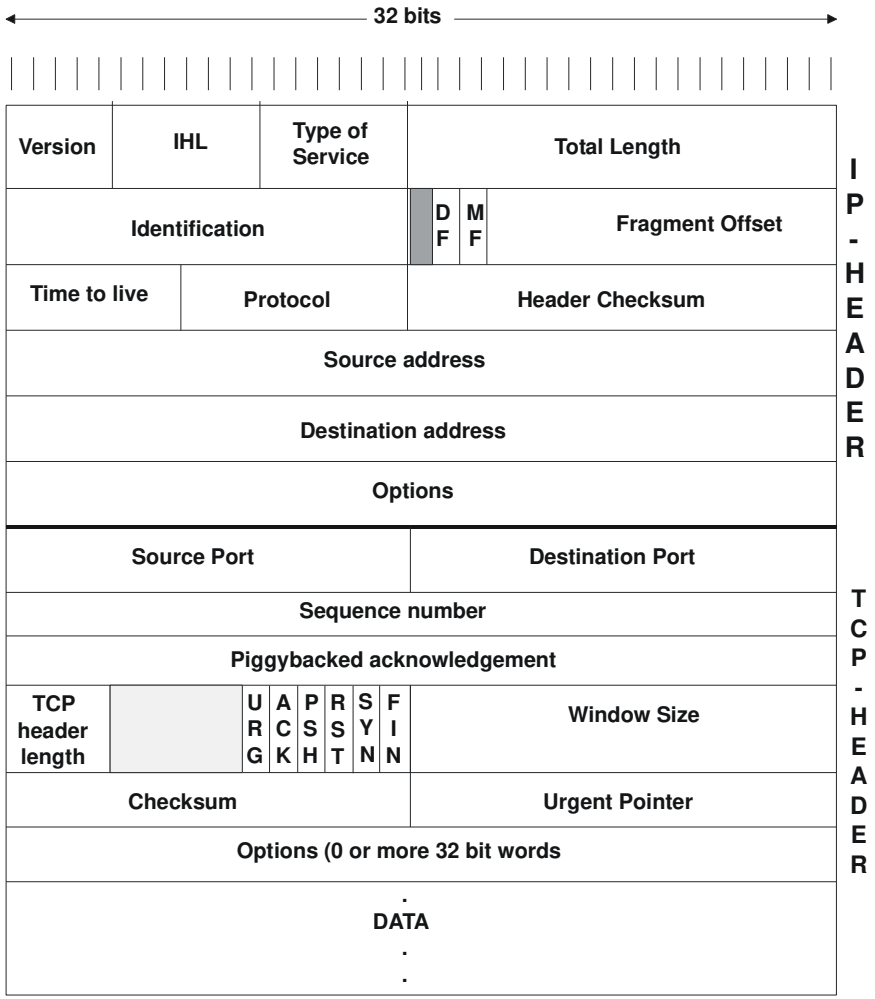
- **codebits** (Flags)
  - URG: urgent pointer Feld ist gültig (“in Benutzung“)
  - ACK: acknowledgement Feld ist gültig (“in Benutzung“)
  - PSH: (push) Der Empfänger soll die Daten der Anwendung so schnell wie möglich zur Verfügung stellen.
  - RST: Reset der Verbindung
  - SYN: Synchronisation der initialen Sequenznummern bei Verbindungsaufbau
  - FIN: der Sender hat das Senden seiner Daten beendet.
- **window size:** Anzahl der Bytes, die der Sender dieses Paketes noch entgegen nehmen kann, bis sein Puffer voll ist (Flusskontrolle)

# Datenfelder im TCP-Header (4)

- **checksum:** Die Prüfsumme wird über das gesamte Fragment berechnet, inklusive TCP-Header. Es wird ein spaltenweises EXOR über die „senkrecht untereinander geschriebenen“ 16-Bit-Worte des Pakets berechnet (wie bei UDP).
- **urgent pointer:** Der “urgent pointer“ identifiziert das letzte Byte im Datenteil, welches mit besonderer Priorität bearbeitet werden sollte. Die danach folgenden Daten haben normale Priorität.
- **options:** (werden wir später besprechen)

Der Datenteil eines TCP-Paketes ist optional, ein leeres TCP-Paket kann zum Beispiel als reine Bestätigung empfangener Daten gesendet werden, wenn gerade keine Daten in die Rückrichtung gesendet werden müssen.

# TCP/IP: Format des gesamten Headers

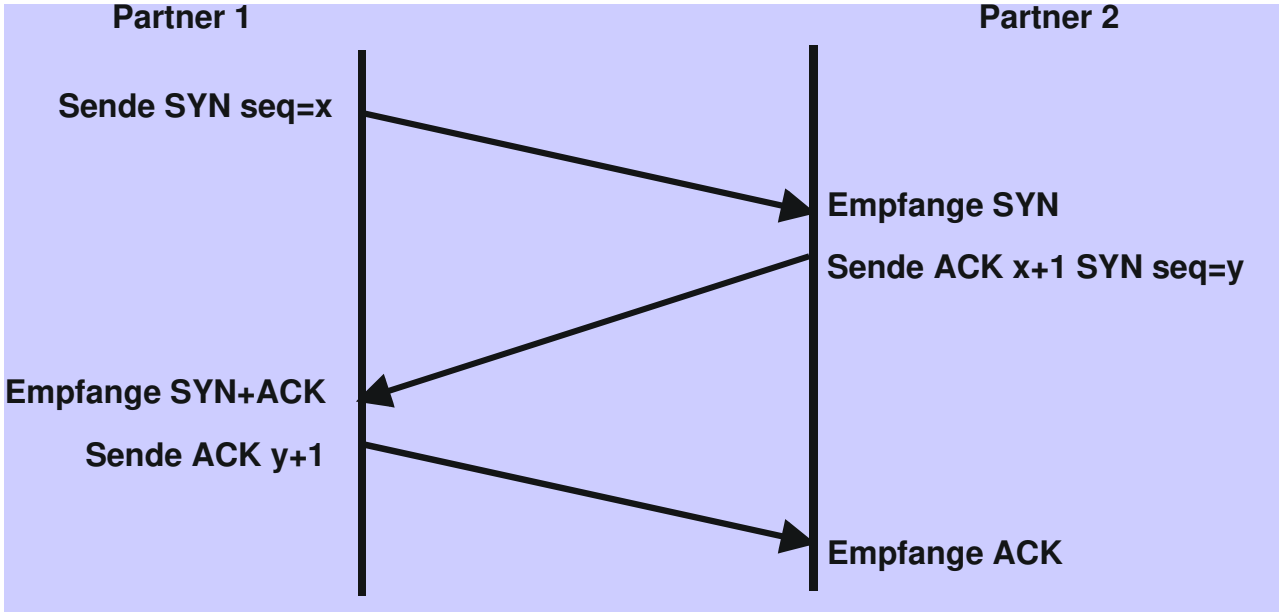


# Wichtige Funktionen in TCP

- Positive Bestätigung ("positive acknowledgement or retransmission„, **PAR**)
- Übertragungswiederholung nach Ablauf einer Wartezeit (Timeout beim Sender). Keine NACKs!
- „Sliding Window"-Mechanismus zur Flusskontrolle. Variable Fenstergröße: bei jeder Bestätigung durch den Empfänger wird die ab sofort zu verwendende Fenstergröße mit übertragen.
- Fehlererkennung durch Prüfsumme
- Piggybacking: Kontrollinformation auf dem Rückweg und Daten können im gleichen TCP-Paket übertragen werden.
- Out-of-Band Data: wichtige Information soll dem Empfänger zugestellt werden, bevor er früher gesendete Daten verarbeitet. Beispiel: Alarm-Meldungen

# TCP-Verbindungsaufbau

**Three-Way-Handshake:** Verbindungsaufbau durch drei Pakete:




Beim Verbindungsaufbau werden auch die initialen Sequenznummern beider Seiten (beider Richtungen) ausgetauscht und bestätigt.

# Three-Way-Handshake

- Das SYN-Flag zeigt an, dass die Sequenznummern synchronisiert werden sollen.
- SYN „kostet“ ein Byte bezüglich der Sequenznummernvergabe.
- Reine acknowledgements „kosten“ keine Bytes.

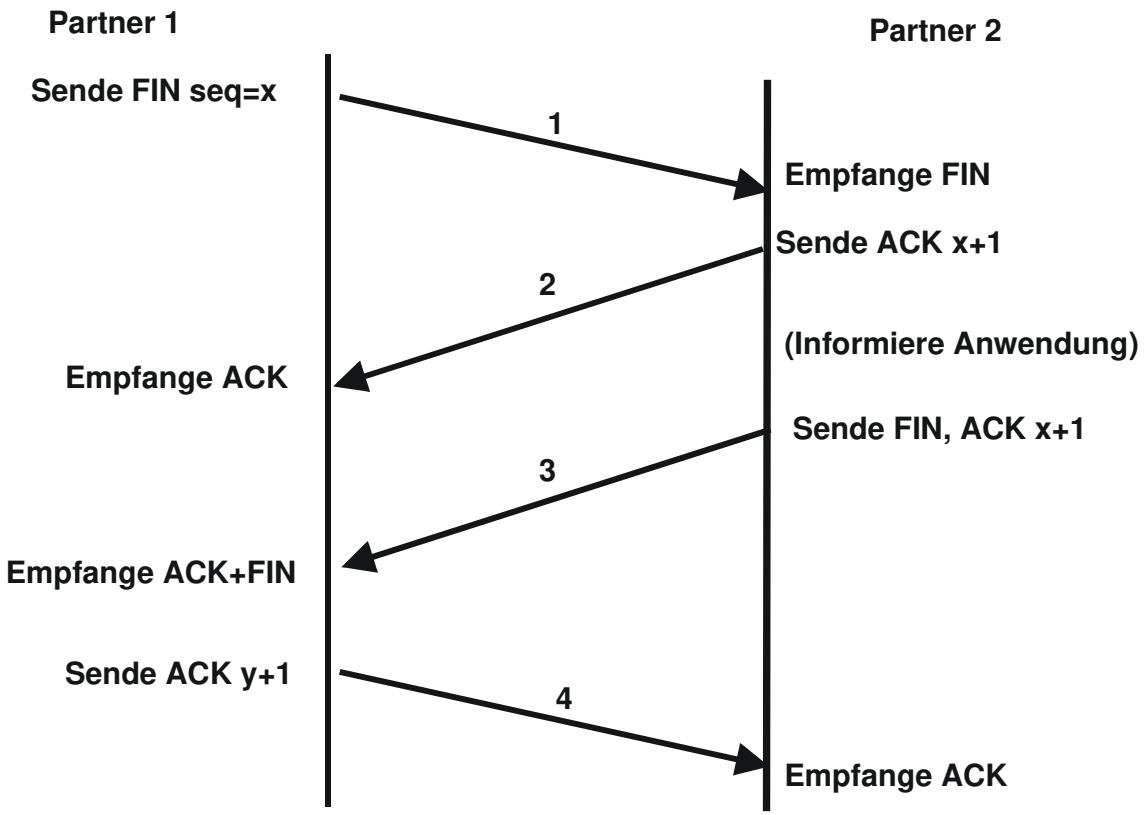
# Timeout beim Verbindungsaufbau

Was passiert, wenn der Kommunikationspartner nicht antwortet?

- Die Übertragung des Paketes wird wiederholt, TCP betrachtet dies als gewöhnlichen Paketverlust.
- Nach einer festen Zeit (timeout) wird der Verbindungsversuch abgebrochen und die Anwendung informiert.

# TCP-Verbindungsabbau (1)

Geordneter Verbindungsabbau durch vier Pakete:



# TCP-Verbindungsabbau (2)

Abbau einer Verbindung durch zweimal „half-close“:

- Da die TCP-Verbindung bidirektional ist, sollten prinzipiell beide Richtungen getrennt voneinander beendet werden.
- Wer seine Senderolle beenden möchte, setzt das FIN-Flag.
- FIN „kostet“ ein Byte und wird daher durch ein acknowledgement bestätigt.
- Der andere Teilnehmer könnte weiter senden - jedoch sieht man in der Praxis fast immer das Verhalten, dass der andere Teilnehmer als Reaktion auch seine Senderolle beendet.

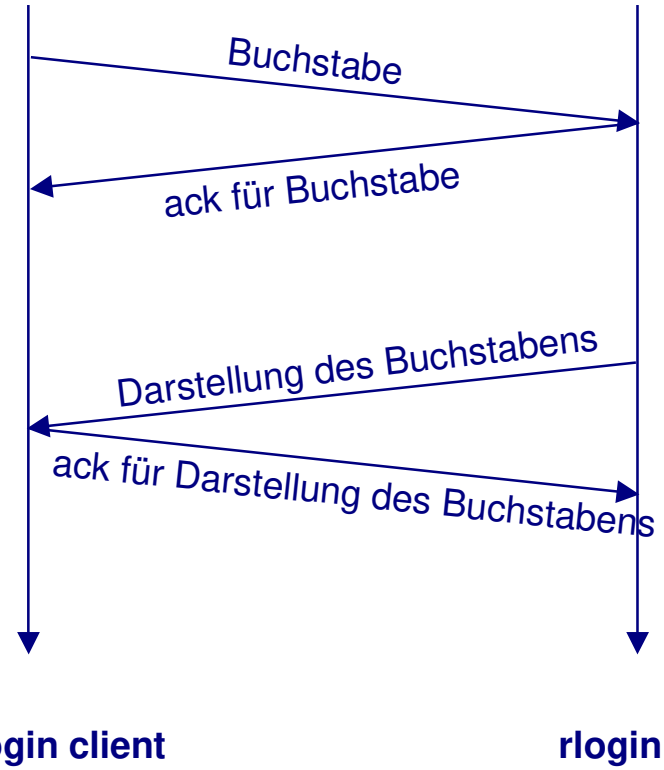
# TCP Reset

Ein Paket, bei dem das RST-Bit im TCP-Header gesetzt ist, terminiert die Verbindung in Form eines „abortive release“ (im Gegensatz zum „orderly release“ mit FIN):

- Alle Daten, die beim Sender gepuffert waren, werden verworfen, und das Reset-Paket wird sofort gesendet. Die Verbindung ist damit aus Sicht des RST-Senders sofort geschlossen.
- Beim Reset können Daten verloren gehen (das passiert beim Verbindungsabbau mit FIN nicht).
- Der Empfänger eines RST-Pakets meldet dies der Anwendung und terminiert die Verbindung sofort.

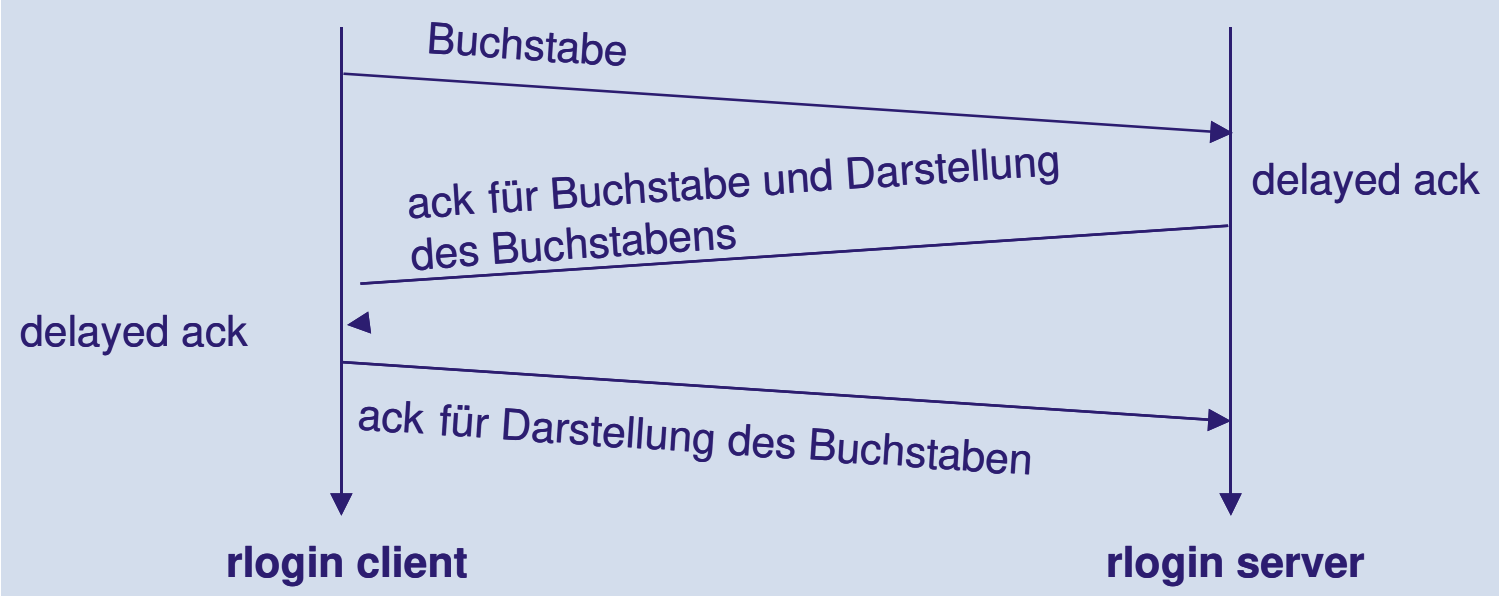
# Typische Datenflüsse mit TCP

Datenflussbeispiel für eine interaktive Anwendung



# Delayed Acknowledgements (1)

Um zu verhindern, dass überflüssige TCP-Pakete gesendet werden, die nur ein ACK beinhalten, wird das Senden von ACKs häufig verzögert:



# Delayed Acknowledgements (2)

- Ein ACK wird bei “delayed acknowledgements“ in der Regel um maximal 200 ms verzögert.
- Während dieser Zeit können Daten, die in der Gegenrichtung gesendet werden, das ACK „Huckepack“ (engl. piggyback) mitnehmen, dies spart die Übertragung eines separaten ACK-Pakets.
- Werden innerhalb dieser Zeit keine Daten gesendet, so wird ein reines ACK-Paket ohne Daten übertragen.
- Der 200-ms-Timer wird nicht für jedes Paket aufgezogen, sondern läuft global, d.h. alle 200 ms werden alle ACKs verschickt, die noch offen sind. Der Timer ist aber natürlich verbindungsbezogen, d. h., für jede TCP-Verbindung und jede Senderichtung gibt es einen eigenen Timer.

# Massendatentransfer

Was passiert, wenn man große Datenmengen per TCP überträgt (bulk data flow)? Wann wird dann ein ACK gesendet?

- Bisher: delayed ACK nach 200 ms
- Das würde zu viele unbestätigte Pakete im Transit verursachen, wenn die Datenrate hoch ist (bulk data flow).
- Daher wird beim Massendatentransfer jedes zweite Paket sofort bestätigt, auch wenn der 200-ms-Timer noch nicht abgelaufen ist.

# Fehlersicherung in TCP

- TCP unterteilt den Bytestrom in Einheiten, die jeweils in einem IP-Paket übertragen werden. Diese Einheiten heißen **Segmente**.
- Nachdem TCP ein Segment per IP losgeschickt hat, wird ein Timer für dieses Segment gestartet.
- Wenn keine Bestätigung über den erfolgreichen Empfang dieses Segments innerhalb der Timer-Laufzeit eintrifft, wird die Übertragung wiederholt.
- Der Timer passt sich an die “normale“ Round-Trip-Time der Verbindung dynamisch an.
- Wenn ein TCP-Empfänger ein fehlerfreies Segment vom Sender erhält, schickt er eine Bestätigung über den erfolgreichen Empfang an den Sender.
- Der Empfänger puffert weitere Segmente, die nach einem Fehler korrekt empfangen wurden, bis der Fehler durch Übertragungswiederholung repariert ist (“go-back-n“ mit Pufferung).
- Negative ACKs (NACKs) werden nicht verschickt!

# Fehlererkennung und -behebung

- TCP berechnet eine Prüfsumme (checksum) über das gesamte Segment. Die Berechnung erfolgt mit demselben Algorithmus wie bei UDP. Falls die Prüfsumme beim Empfänger einen Fehler signalisiert, wird das Segment nicht bestätigt. Das führt zum Ablaufen der Zeitschranke beim Sender und als Folge davon zur Übertragungswiederholung. Der Sender wiederholt das Senden nach dem „go-back-n“-Verfahren.
- Wenn Segmente außer der Reihe von IP ausgeliefert werden, stellt TCP die richtige Reihenfolge wieder her.
- Wenn IP-Datagramme im Netz verdoppelt werden, dann filtert TCP die Duplikate heraus.

# Flusskontrolle und Überlastkontrolle in TCP

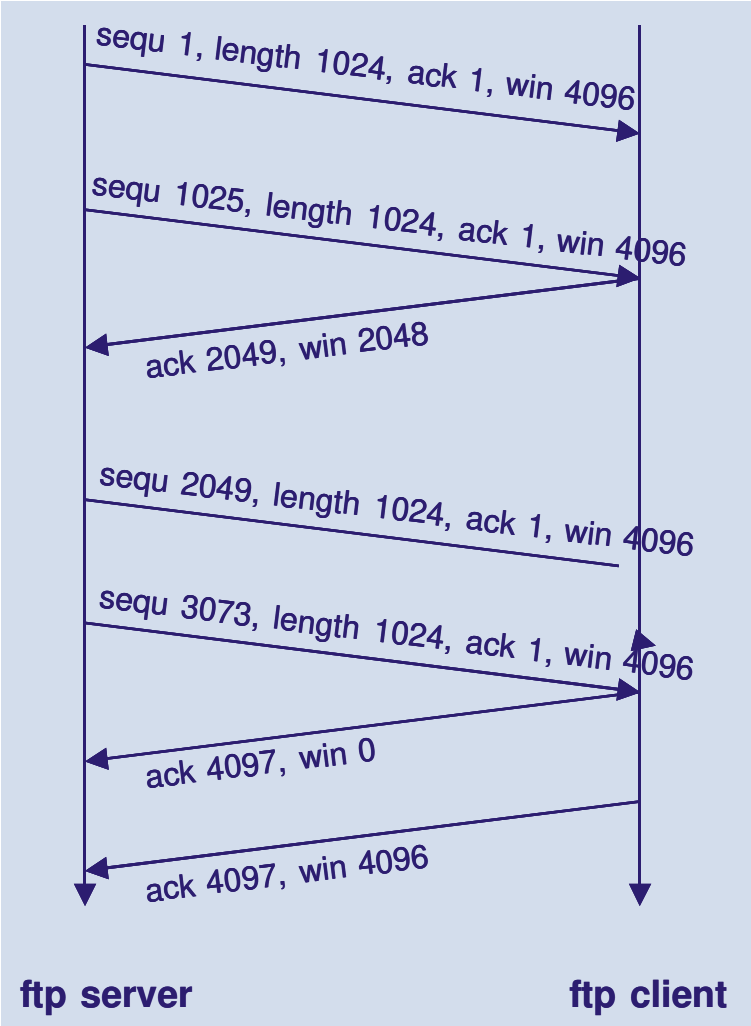
## Flusskontrolle

TCP verwendet einen Schiebefenster-Mechanismus (sliding window) zur Flusskontrolle. Die Fenstergröße wird, wie stets in TCP, in Bytes (nicht in Paketen!) ausgedrückt.

Die Größe des Schiebefenster wird als Flusskontrollparameter „window size“ vom Empfänger an den Sender geschickt.

Die Größe des Fensters kann während der Verbindung geändert werden. Wenn der Empfänger beispielsweise nur noch wenig Pufferplatz hat, wird sie reduziert.

# Schneller Sender und langsamer Empfänger (1)

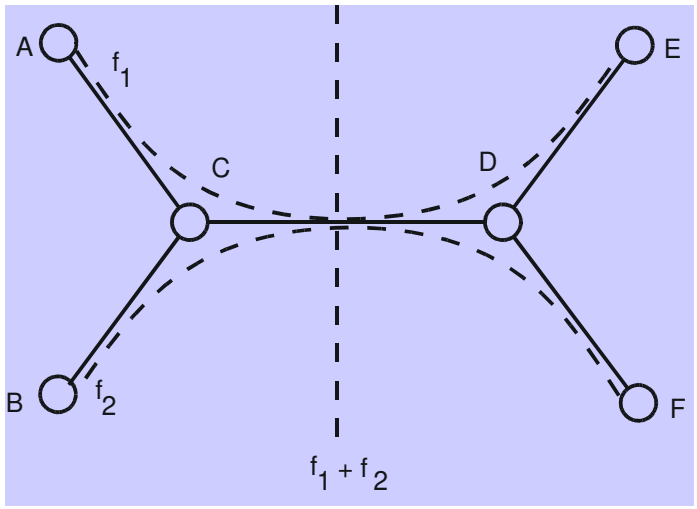


# Schneller Sender und langsamer Empfänger (2)

- Der Sender überträgt die Daten schneller, als sie der Empfänger aus dem Puffer lesen und an die höheren Schichten weiter geben kann.
- Der Puffer des Empfängers läuft voll, er signalisiert dies mit der Fenstergröße 0. Dies ist eine Erweiterung zum eigentlichen Sliding-Window-Mechanismus, welche es erlaubt, Pakete bezüglich der Fehlersicherung zu bestätigen, ohne dem Sender das Recht zu geben, weitere Pakete zu senden.
- Erst wenn der Puffer beim Empfänger wieder freien Platz hat, wird dieser freie Platz in einem weiteren ACK als Fenstergröße dem Sender mitgeteilt.

# Überlastkontrolle (Congestion Control)

Problem: Wenn alle Sender im Netz immer so viele Pakete losschicken, wie bei ihren Empfängern in die Puffer passen, dann kann es zur Überlast (Verstopfung, congestion) im Inneren des Netzes kommen.




Wenn alle Verbindungen die gleiche Bandbreite haben und sowohl A als auch B mit der vollen Bandbreite der Verbindung senden, dann kommt es zu einer Überlastung im Inneren des Netzes (**congestion**). TCP-Verbindungen erkennen dies und regeln **freiwillig** die Bandbreite herunter!

# Congestion Window

- Um Congestion zu verhindern, wird beim Sender ein zusätzlicher Parameter **Congestion Window** (cwnd) mitgeführt.
- cwnd wird wie das vom Empfänger mitgeteilte flow control window in Bytes geführt.
- Ein Sender darf immer nur das MINIMUM aus (cwnd, Flusskontroll-Window) an Daten senden.
- Es besteht keine Notwendigkeit, cnwd zwischen den Partnern zu übertragen!

# Überlastkontrolle im eingeschwungenen Zustand

**Problem**

Wie stellt der Sender die richtige Größe für das Congestion Window fest?

**Lösung**

Er versucht, das Fenster schrittweise zu vergrößern, bis Verstopfung eintritt. Dann verkleinert er es wieder.

Solange keine Pakete verloren gehen, wird bei jedem ACK cwnd um 1/cwnd Segmente erhöht. Pro Round-Trip-Time vergrößert sich das Congestion Window also um ca. ein Segment ("*additive increase*").

Da es keine spezifischen Meldepakete aus dem Inneren des Netzes für Verstopfung gibt, interpretiert TCP jeden Paketverlust der Verbindung als Anzeichen für Überlast!

Es gibt zwei unterschiedliche Reaktionen auf Segmentverluste, wie nachfolgend beschrieben.

# Anzeichen für Überlast

- **Triple Duplicate Acknowledgement** (TDACK): Wenn ein Segment verloren geht, aber nachfolgende Segmente ankommen, erhält der Sender mehrere ACKs mit derselben Sequenznummer. Nach dem dritten „Duplicate ACK“ (also vier ACKs insgesamt für dasselbe Segment) wird das fehlende Paket erneut übertragen, ohne dass der Timeout für das Paket abgewartet wird (“fast retransmit“). Der Sender interpretiert ein TDACK als „leichte“ Überlast und reduziert cwnd **auf die Hälfte** der ursprünglichen Größe (“*multiplicative decrease*“).
- **Timeout**: Wenn nach einem verlorengegangenen Paket auch keine Folgepakete mehr ankommen, kommt es nicht zu einem TDACK, sondern zu einem Timeout für das verlorene Paket. Der Sender interpretiert das als schwere Überlast und reduziert cwnd **auf ein Segment**.

# Wahl des Timeout-Wertes

**Frage:** Wie groß sollte der Timeout-Wert gewählt werden?

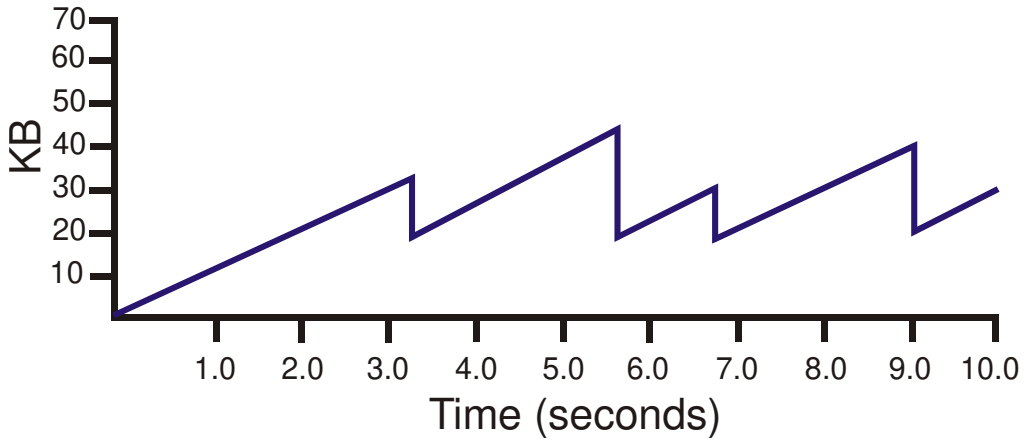
- Auf jeden Fall größer als eine Round-Trip Time (RTT)
- RTT kann variieren  Sicherheitszuschlag in Abhängigkeit der RTT Varianz
  - EstimatedRTT = (1-x) * EstimatedRTT + x * SampleRTT
  - Deviation = (1-x) * Deviation + x * abs(SampleRTT - EstimatedRTT)
  - Timeout = EstimatedRTT + 4 * Deviation
- Ein guter gewählter Timeout-Wert ist wichtig für hohen Datendurchsatz in Netzen, in denen häufig Paketverluste auftreten.

# Sägezahnkurve des TCP-Durchsatzes

Der Algorithmus *Additive Increase, Multiplicative Decrease* führt dazu, dass der tatsächliche Durchsatz einer TCP-Verbindung im eingeschwungenen Zustand eine Sägezahnkurve ergibt. Aus regelungstechnischer Sicht ist das Verfahren stabil, d. h. es schwingt sich nicht auf.

Beobachtung: Die Fläche unter der Kurve entspricht der TCP-Datenrate!

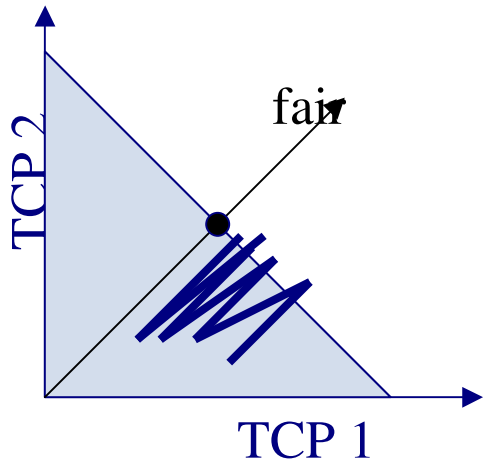
Ein Beispiel zeigt die unten stehende Abbildung.



# TCP Fairness

Wenn sich *n* TCP-Ströme einen Link teilen, sollte jeder einen Durchsatz von ca. 1/*n* der verfügbaren Bandbreite in Anspruch nehmen.

Beispiel: Zwei TCP-Ströme laufen über einen gemeinsamen Link:




Die Datenraten tendieren gegen den Punkt der fairen Kapazitätsverteilung unter voller Ausnutzung der vorhandenen Bandbreite.

# Slow-Start (1)

**Problem:**

Wenn eine TCP-Verbindung neu aufgebaut wird, dauert es sehr lange, bis der Sender die optimale Bitrate erreicht, da sich mit jeder Round-Trip-Time *cwnd* nur um eine Paketgröße (MSS = max. segment size) erhöht.

**Lösung:**

Der **Slow-Start-Algorithmus** von Van Jacobsen

- *cwnd* wird mit der MSS des Empfängers initialisiert
- slow start threshold (*ssthresh*) wird mit 65353 initialisiert
- Pro Round-Trip-Time tue folgendes:
  - Wenn keine Verluste und *cwnd* < *ssthresh*:
    **Slow-Start-Phase:** erhöhe *cwnd* um MSS für jedes empfangene ACK (dies ist exponentiell!)
  - Wenn Verluste oder *cwnd* >= *ssthresh:*
    **Ende der Slow-Start-Phase:** normaler AIMD-Algorithmus setzt ein.

# Slow-Start (2)

Ein TCP-Slow-Start findet ebenfalls nach einem Timeout statt (Anzeichen für schwere Verstopfung). Der *ssthresh*-Wert wird halbiert, dann beginnt eine neue Slow-Start-Phase.

**Anmerkung**

Dieser Algorithmus heißt aus historischen Gründen „slow start“, obwohl es aus heutiger Sicht eigentlich ein “Quick-Start-Algorithmus“ ist. Denn bevor er in TCP eingebaut wurde, begann jede neue Verbindung mit einer Fenstergröße, die dem **Flusskontroll**fenster des Empfängers entsprach. Dies führte häufig sofort zu Verstopfung und Paketverlusten.

# Slow-Start: Beispiel für einen Verlauf

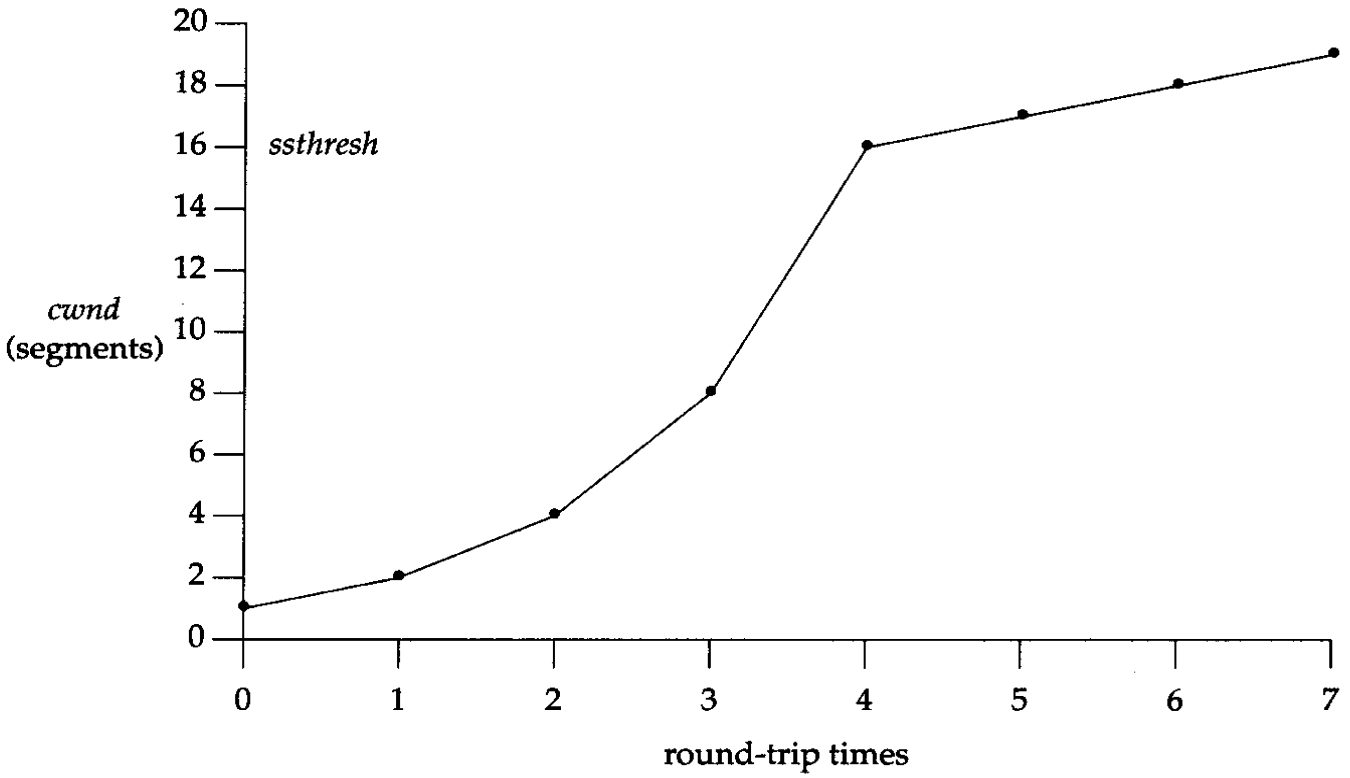



**Figure 21.8** Visualization of slow start and congestion avoidance.

# TCP-Programmierschnittstelle (API)

Die Programmierung von TCP erfolgt mit dem Konzept der **Sockets**. Ein Socket ist die programmiertechnische Realisierung eines Ports.

- Der Server wartet auf einem wohl definierten Port auf Verbindungswünsche von Clients.
- Ein Client verbindet sich mit dem Server .
- Nachdem die Verbindung hergestellt ist, kann der Server für diese Verbindung einen Thread (leichtgewichtigen Prozess) anlegen, der dann die weiterhin eingehenden Datenpakete der Verbindung bearbeitet.
- Wenn er keinen neuen Thread anlegt, werden die Verbindungswünsche sequentiell bearbeitet (selten).

# Zusammenfassung TCP

## Vorteile von TCP

- Gesicherte Datenübertragung
- Effiziente Datenübertragung trotz Komplexität des Protokolls (bis zu 100 MBit/s auf Standard-Maschinen experimentell nachgewiesen)
- Einsetzbar im LAN- und WAN-Bereich
- Für geringe Datenraten (z. B. interaktives Terminal) und hohe Datenraten (z. B. Dateitransfer) gut verwendbar

## Nachteile gegenüber UDP

- Höherer Ressourcenbedarf (Zwischenspeicherung, Zustandsinformationen bei Sender und Empfänger, viele Timer)
- Verbindungsaufbau und -abbau auch bei kurzen Datenübertragungen notwendig
- Multicast nicht möglich