TECHNISCHE UNIVERSITÄT DRESDEN

Fakultät Informatik | Institut Systemarchitektur | Professur Rechnernetze

WS 2016/2017
LV Rechnernetzpraxis

9. Transportprotokolle

Dr. rer.nat. D. Gütter

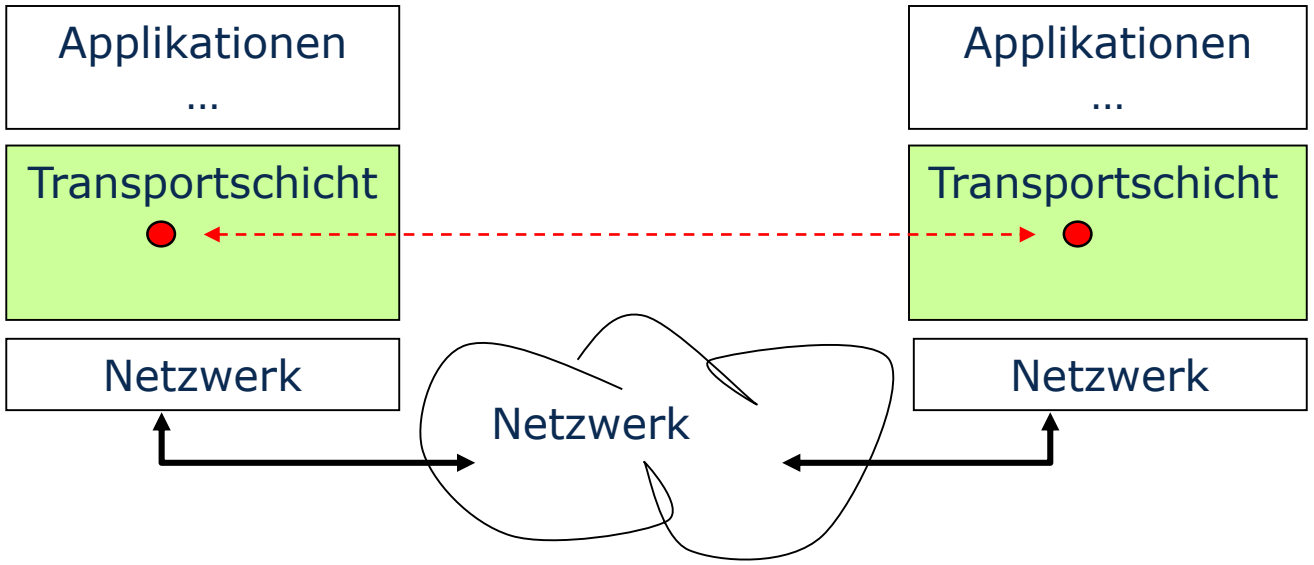
Mail: Dietbert.Guetter@tu-dresden.de
WWW: http://www.guetter-web.de/education/rnp.htm

05.01.2017

Aufgaben der Transportschicht

- Entlastung der verarbeitungsorientierten Schichten von **allen** Übertragungsproblemen

- globale Adressierung von Applikationsprozessen
- zuverlässige Ende-zu-Ende Datenübertragung, auch bei unzuverlässiger Netzwerken
- evtl. Garantie von Dienstgüte-Eigenschaften (QoS)



Transportprotokolle

- **ISO/OSI** 5 Transportprotokollklassen

  TP0 – geringe Funktionalität, nur geeignet für sehr gute Netzwerke
  …
  TP4 – Fehlerkontrolle, Flußsteuerung, …, Applikationsmultiplexing
  geeignet auch für unzuverlässige Netzwerke

- **Internet** 2 Transportprotokolle

  UDP – verbindungslos,
  geringe Dienstqualität (wie IP), Multiplexnutzung möglich

  TCP – verbindungsorientiert,
  Fehlerkontrolle, Flußsteuerung, …, Applikationsmultiplexing
  geeignet auch für unzuverlässige Netzwerke

- …

→Trend zu durchgängiger Nutzung von Internetprotokollen

Internet - Socketschnittstelle

Socket Kommunikationsendpunkt
für bidirektionalen Datenaustausch über T-Protokolle TCP, UDP
sowie über IP (Raw Sockets)

Adressierung über Port-Nr. und IP-Adresse

Plattform 1981, Berkeley UNIX BSD, entwickelt für Programmiersprache C

inzwischen auf allen Plattformen in vielen Sprachen verfügbar
z.B. „WinSock" für Windows

Schnittstelle angelehnt an UNIX-Filesystem, Client-/Server-Modell

Server wartet	→ eigene Socketadresse bekannt Partneradresse offen
Client initiativ	→ eigene Socketadresse bekannt → Server-Socketadresse bekannt

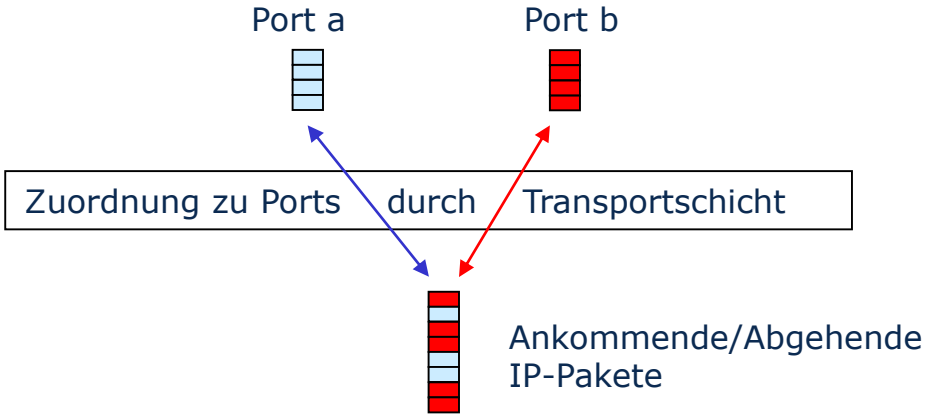
Adressierung bei Internet-Transportprotokollen

Socketadresse

Rechner wird adressiert über IP-Adresse
Prozeß wird adressiert über Ports

Port := Nachrichtenübergabestelle auf einem Rechner
dargestellt als 16-Bit-Zahl
(TCP-Ports unabhängig von UDP-Ports)

entspricht einer speziellen Mailbox
zur Kommunikation zwischen Nutzer und Transportschicht



Applikationen – Zuordnung T-Protokoll und Port

Client-/Server-Kommunikation

Client ergreift Initiative → Serverport muß bekannt sein
→ Client-Port frei wählbar

Bereich	Nutzung	Vergabe
0	reserviert	Betriebssystem vergibt selbständig freien Port
1 … 1023	well known ports	Nutzung nur durch Superuser (root) Vergabe nach Antrag durch IANA
1024 … 49151	registered ports	Nutzung frei Registrierte Ports (IANA)
49152 … 65535	ephemeral ports	Nutzung frei keine Registrierung

Applikationen – Beispiele der Zuordnung

→ http://www.iana.org/assignments/port-numbers

Applikationsprotokoll	Transportprotokoll	Serverport
FTP-Data	UDP,TCP	20
FTP-Control	UDP,TCP	21
Telnet	TCP	23
SMTP	TCP	25
DNS	UDP,TCP	53
DHCP	UDP	67
HTTP	TCP	80
POP3	TCP/UDP	110
SNMP	UDP,TCP	161, 162
BGP	TCP	179
RIP	UDP	520
MySQL	TCP/UDP	3306

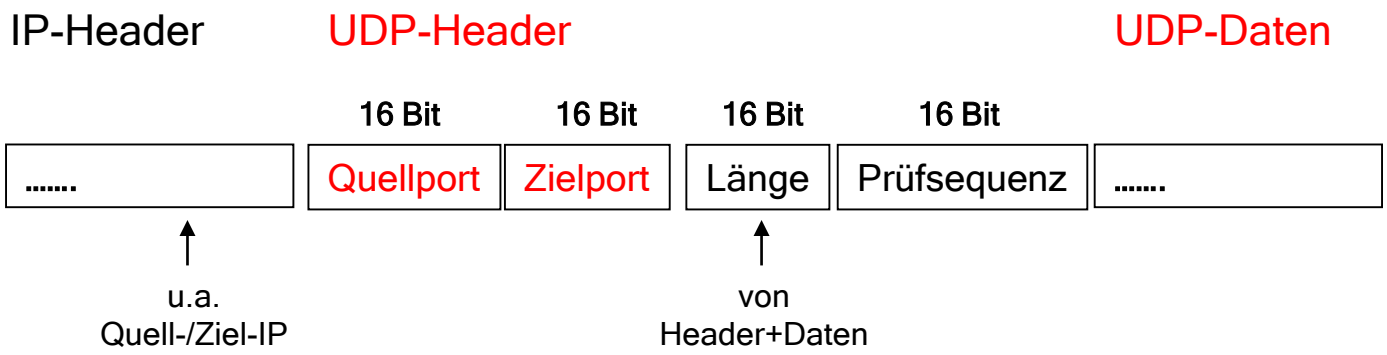
UDP (User Datagram Protocol)

RFC 768

Transportprotokoll,
verbindungslos, keine Fehlerkorrektur, keine Flußsteuerung

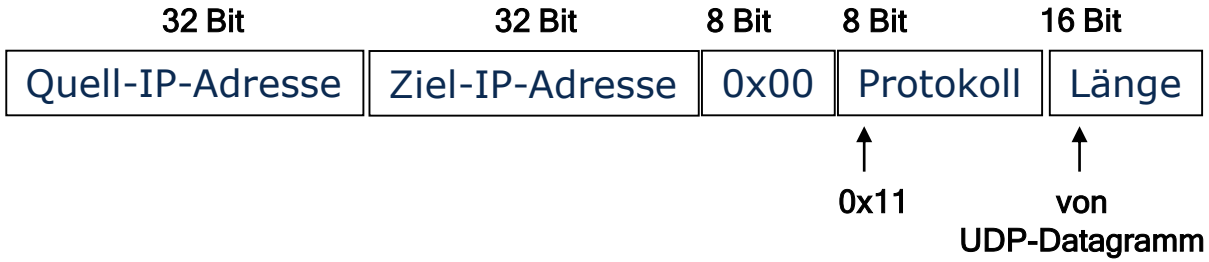
→ geringer Overhead
→ keine Verzögerung durch Verbindungsaufbau
→ Betriebssystembelastung gering
→ geringe Zuverlässigkeit

UDP-Protokoll-Dateneinheiten werden als IP-Pakete versendet.



UDP – Prüfsequenz

interne Bildung eines Pseudoheaders, wird nicht (!) übertragen



UDP-Prüfsequenz wird berechnet über

- Pseudoheader
- UDP-Header (Prüfsequenzfeld mit 0x0000)
- UDP-Daten

Ausnahmen:

Prüfsequenz	= 0	keine Prüfung erfolgt
	= 0xFFFF	Berechnung ergab eigentlich 0x0000

Berechnung wie bei IP (Einerkomplement der Summe von 16-Bit-Worten)

TCP (Transmission Control Protocol)

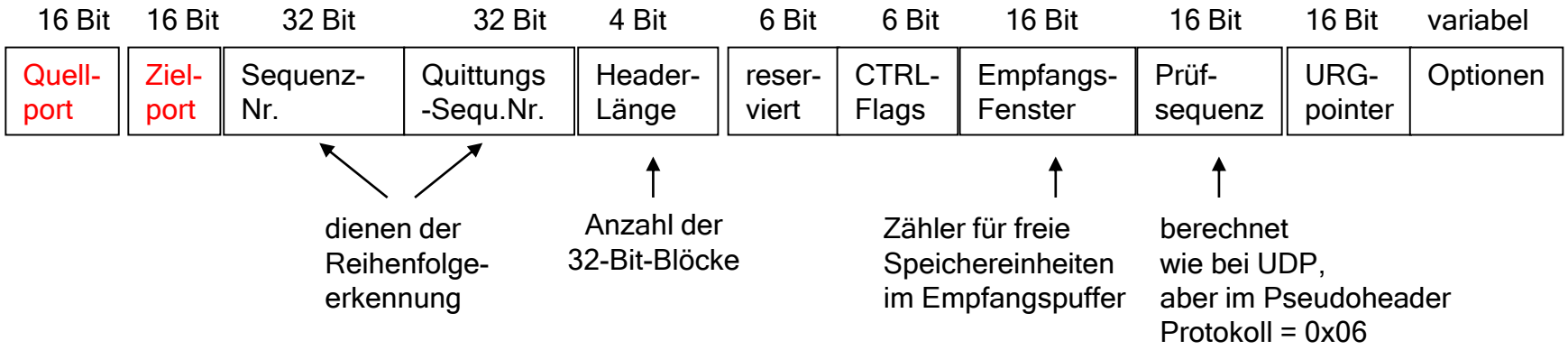
RFC 793

Transportprotokoll,
verbindungsorientiert, Fehlerkorrektur, Flußsteuerung

→ hoher Overhead
→ hohe Zuverlässigkeit

Protokoll-Dateneinheiten (TCP-Segmente) werden als IP-Pakete versendet.

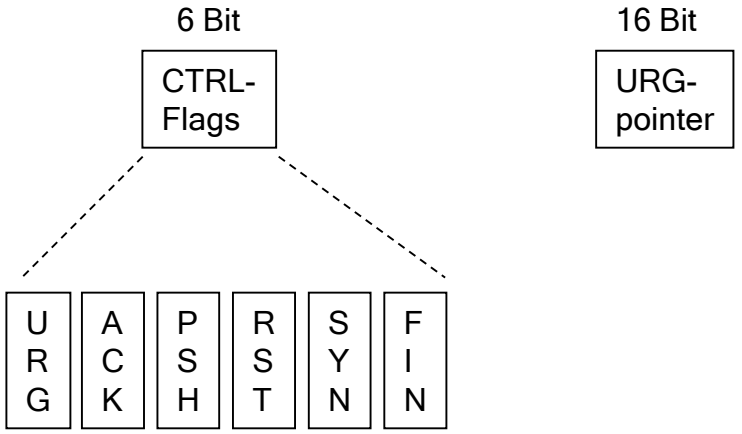
Segment-Header



TCP – Header Control Flags

URG (urgent flag)
aktuelles Byte (s. URG-Pointer)
soll sofort vom Empfänger
bearbeitet werden

ACK (acknowledgement flag)
Quittungs-Nr. ist gültig
(sonst ignorieren)



PSH (push flag)
Zwischenpufferung von Daten verhindern, sofort senden

RST (reset flag)
Verbindung abbrechen

SYN
Verbindungsinitialisierung, Antwort mit SYS+ACK oder RST
Synchronisation der Sequenznummern beim Verbindungsaufbau

FIN (finish flag)
Freigabe der Verbindung, es folgen keine Daten mehr

TCP – Verbindungsaufbau

Client

kennt Server-Socketadresse (Standard)

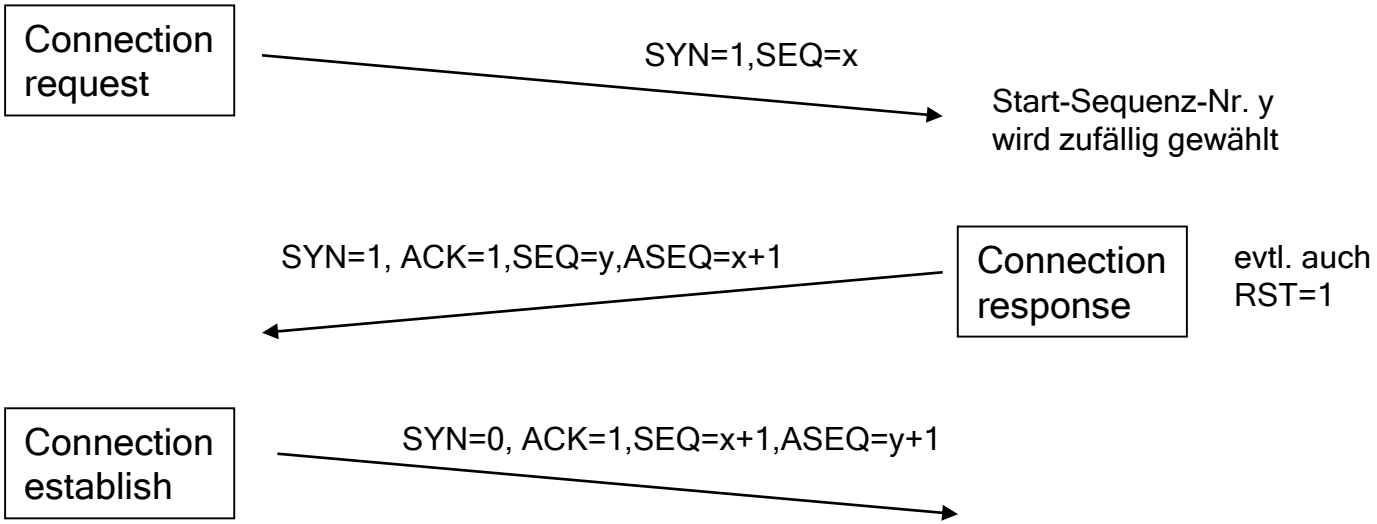
fordert freien Port vom Betriebssystem →Client-Socketadresse

Start-Sequenz-Nr. x wird zufällig gewählt

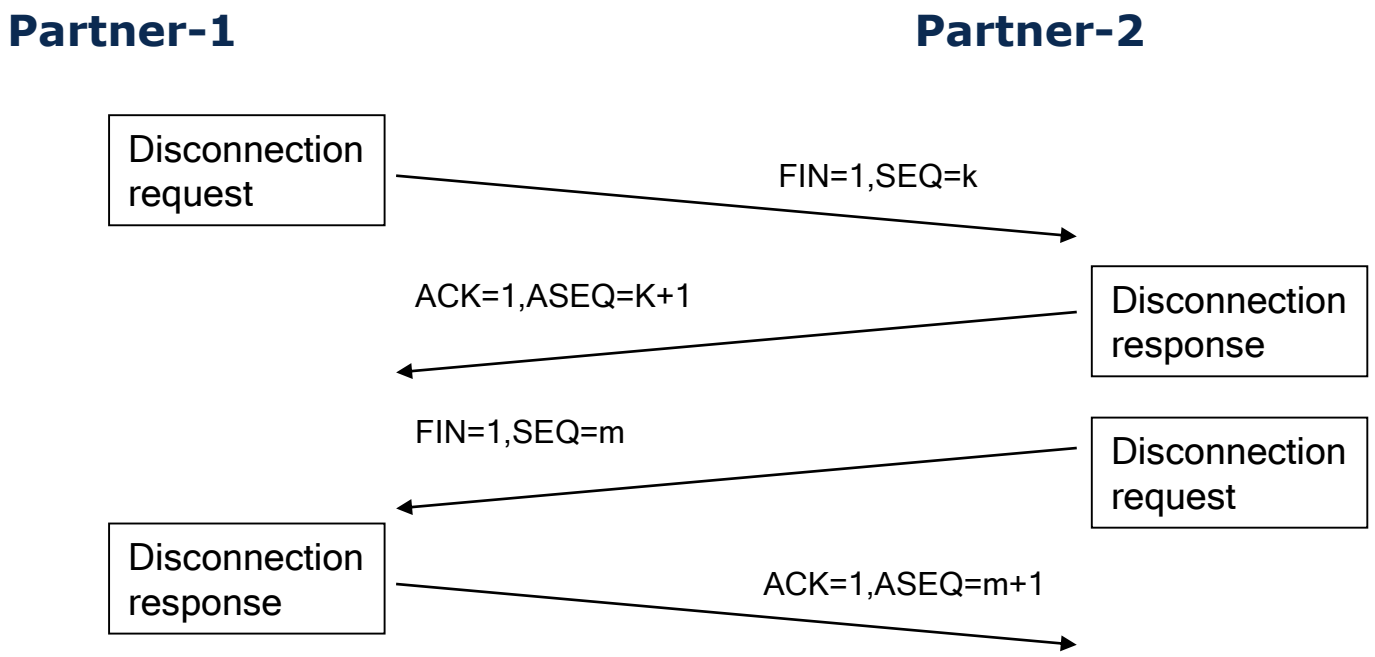
Server

vereinbart mit Betriebsystem die Nutzung des Serverports

wartet auf Nachrichten



TCP – Verbindungsabbau



beide (!) Seiten müssen Verbindung abbauen

evtl.
Zusammenfassung von ACK und FIN zu einer Nachricht bei Partner-2
(3-Wege-Handshake)

TCP – Datenübertragung

TCP behandelt Datenübertragungen ähnlich wie Ein-/Ausgaben bei Dateien

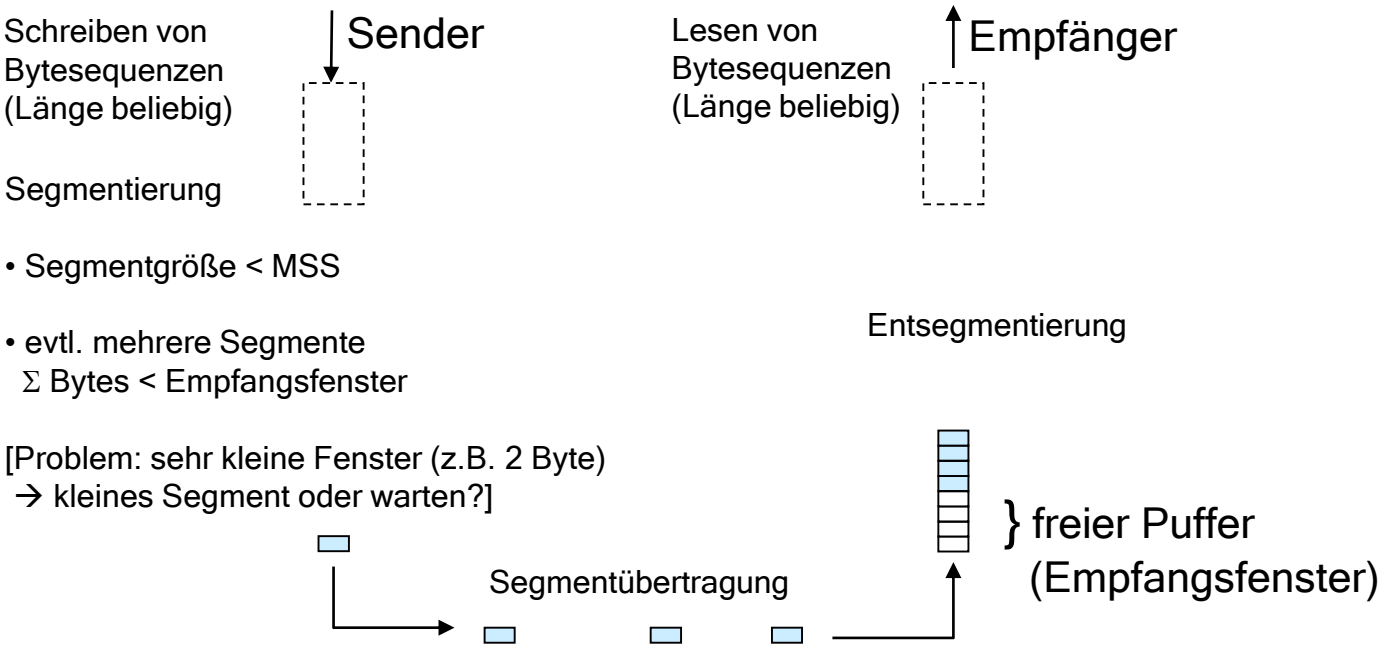
- bidirektionale Übertragung (Duplex)
- physische Blockung der Daten ist transparent
- TCP puffert die Daten so, daß der Durchsatz optimiert wird
- Flußsteuerung
  - Vermeidung von Überlastsituationen
  - gerechte, faire Ressourcennutzung
- nicht immer aus Anwendersicht günstig, z.B.
  - bei Telnet-Kommandoeingaben (^C)
  - bei interaktiver Arbeit
  - bei Verarbeitung von Datensätzen fester Größe

Ausnahmebehandlung möglich über PSH- und URG-Flag

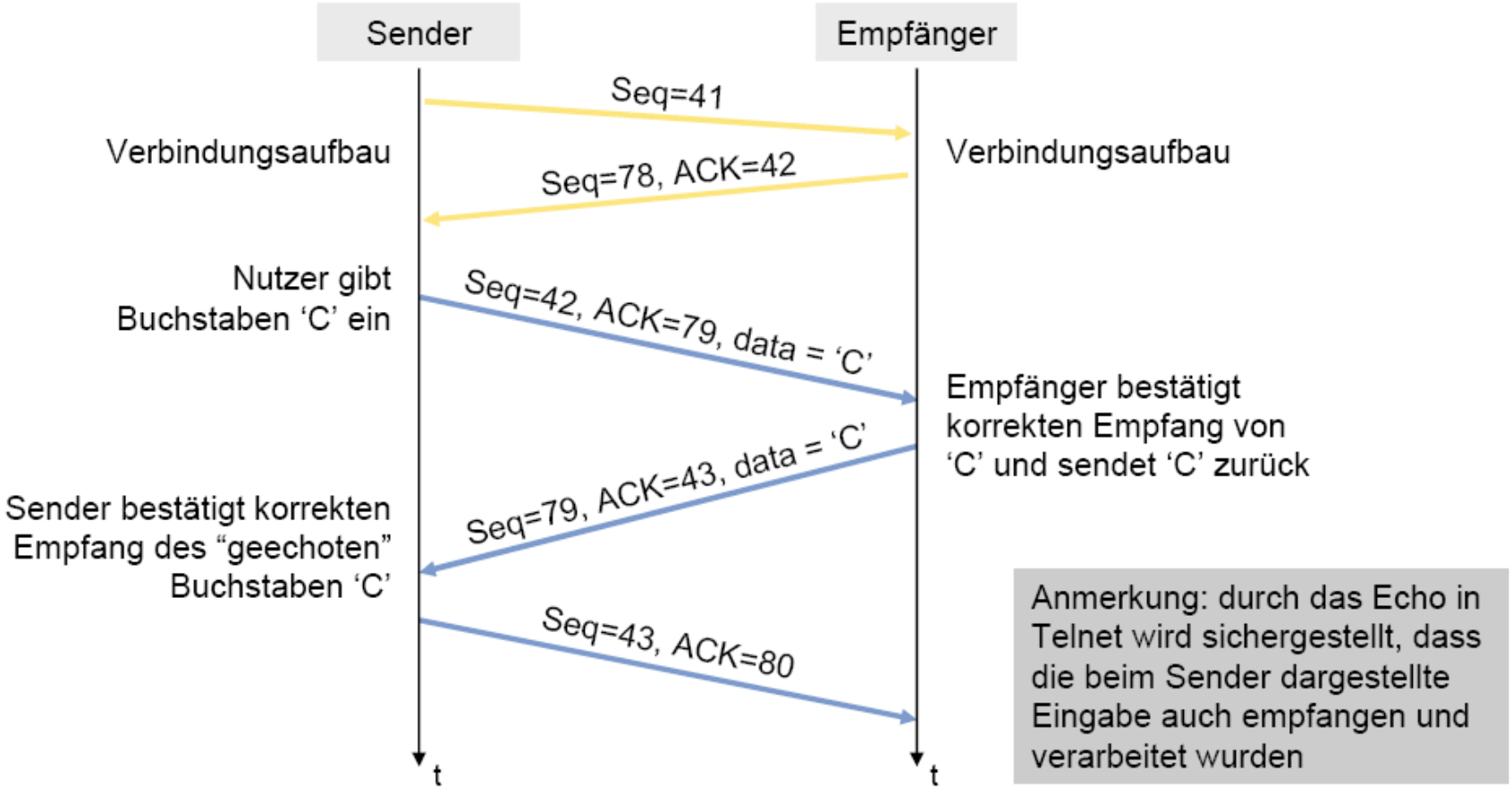
TCP – Flußsteuerung (Flow Control)

- Während Verbindungsaufbau erfolgt Vereinbarung einer maximalen Segmentgröße MSS (Maximum Segment Size)

- Empfangsfenster beschreibt Leerstand des Empfangspuffers.
  ursprünglich: Einheit 1 Byte → Fenster max. 2^{16} Bytes
  RFC 1323: Multiplikation mit Faktor → Fenster max. 2^{30} Bytes
  (TCP Window Scale Option)



TCP – Beispiel zur Datenübertragung (Telnet)



Anmerkung: durch das Echo in Telnet wird sichergestellt, dass die beim Sender dargestellte Eingabe auch empfangen und verarbeitet wurden

TCP – Flußsteuerung (Flow Control)

- Alle TCP-Segmente werden quittiert
- Positive kumulative Quittungen

  ACK=1, ASEQ=anzahl → Quittierung aller Segmente bis Segment$_{anzahl}$

- Empfänger kann Überflutung verhindern
  durch Steuerung der Größe des Empfangsfensters (Flußsteuerung)
- Quittungsempfang wird mit Time-out (Retransmission Timer) überwacht

  Timergröße problematisch (Auslösung nur im Fehlerfall, aber auch schnell)
  Messungen erforderlich für Umlaufzeit (RTT – Round Trip Time)

  Empfehlung: Timer = 2 * EstimatedRTT

- Fehlerursachen — Netzüberlastung, Stau in Routern, ..
- Reaktion im Fehlerfall — → Go-Back-N
  (mit Erweiterungen in Richtung Selective-Repeat)
  → Sendeflußreduktion

TCP – Stauverhinderung (Congestion Control)

Time-Out für Quittungsempfang, evtl. Duplikate

- Stau, lange Verzögerungen
- Router-Überlastung (Pufferüberlauf, …)

→ Netz wahrscheinlich überlastet , Reduktion Senderate sinnvoll

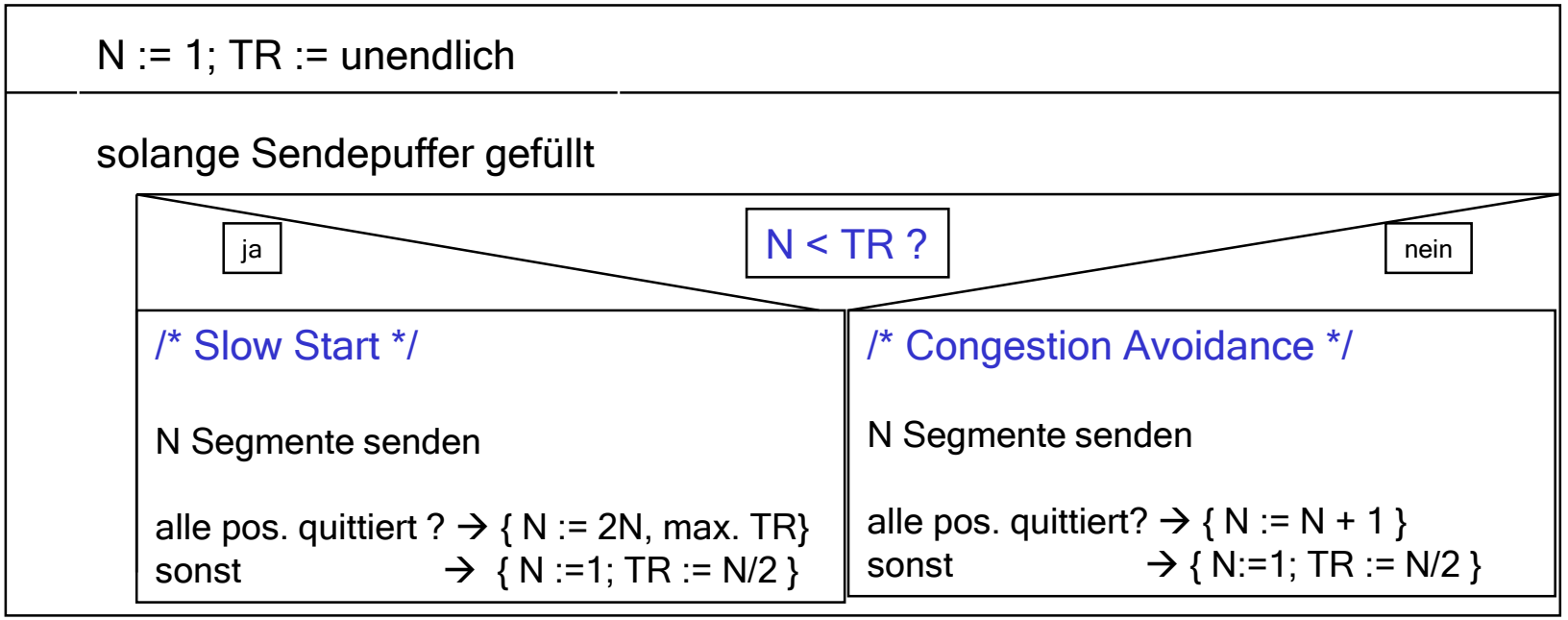
Congestion Control

- Sender führt Sendefenster (Congestion Window)
- Byteanzahl unbestätigter Daten !< min { Sende-/Empfangsfenster }
- Probephase (Annäherung an maximale Senderate)
  Reduktion der Senderate bei aufgetretenem Stau
- Fairness bei Konkurrenz von Verbindungen
  → Endzustand: jede Verbindung gleiche Senderate
  (problematisch, wenn
  eine Anwendung mehrere Verbindungen parallel betreibt)

TCP – Slow Start und Congestion Avoidance

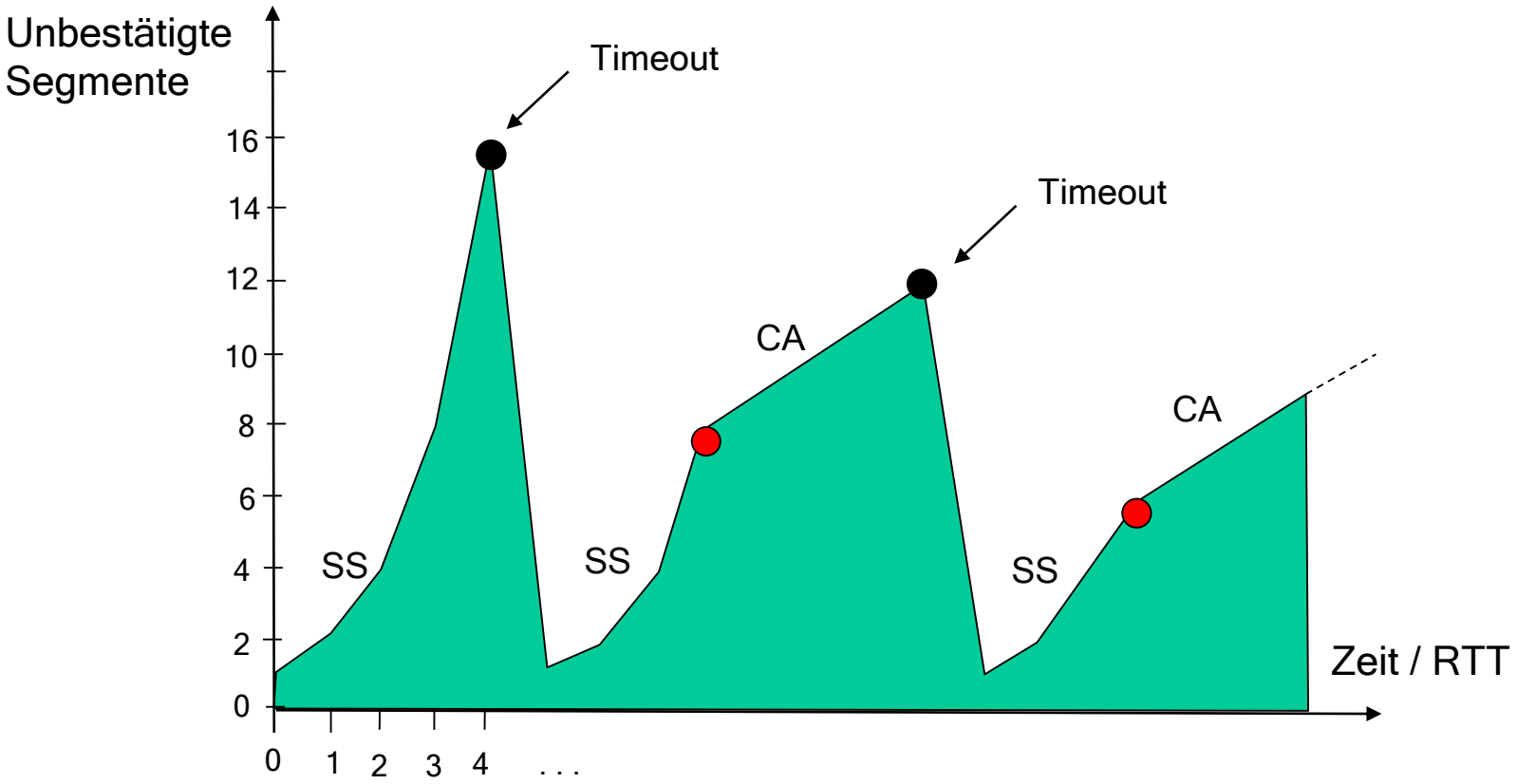
größere Datenmengen werden in Segmenten der Größe MSS übertragen

RTT	Round Trip Time Zeit zwischen Segmentsenden und Quittungsempfang
N	Anzahl unbestätigt sendbarer Segmente (max. RTT/MSS)
SF	Sendefenster (gleich N * MSS)
TR	Schwellwert für N → Algorithmenumschaltung



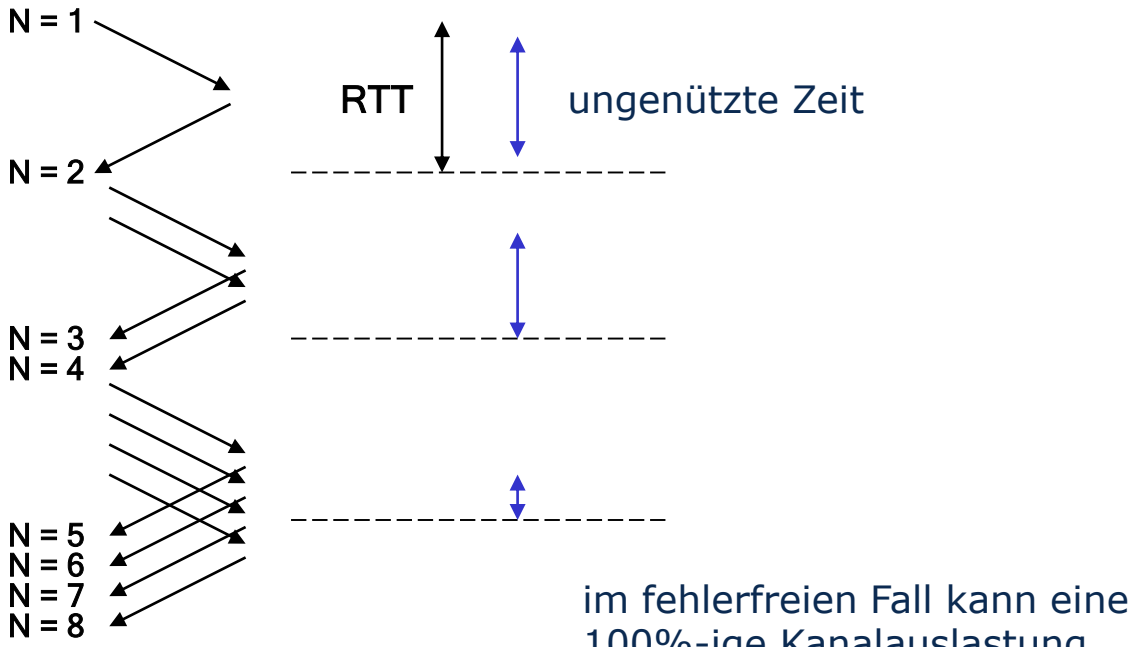
1988 TCP Tahoe – Beispielablauf Sendesteuerung

- exponentieller Anstieg bei Slow Start
- linearer Anstieg bei Congestion Avoidance



TCP – Ablaufdiagramm „Slow Start“

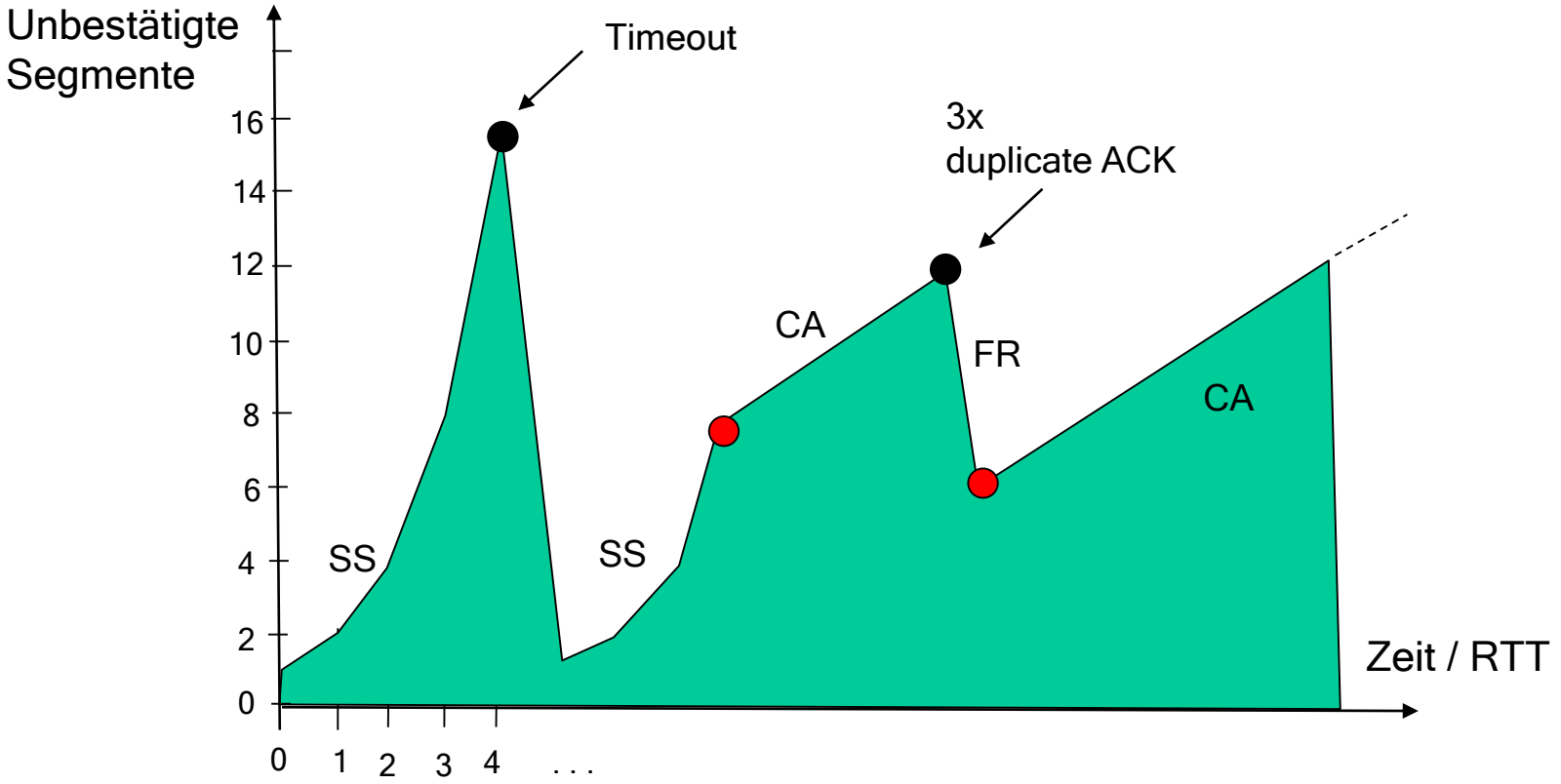
bei jedem Quittungsempfang wird Sendefenster um 1 Einheit erhöht
→ innerhalb jeder RTT eine Sendefensterverdopplung



im fehlerfreien Fall kann eine 100%-ige Kanalauslastung schnell erreicht werden

1990 TCP Reno – Beispielablauf Sendesteuerung

- exponentieller Anstieg (Slow Start)
- linearer Anstieg (Congestion Avoidance)
- Fast Recovery (3x duplicate ACK)



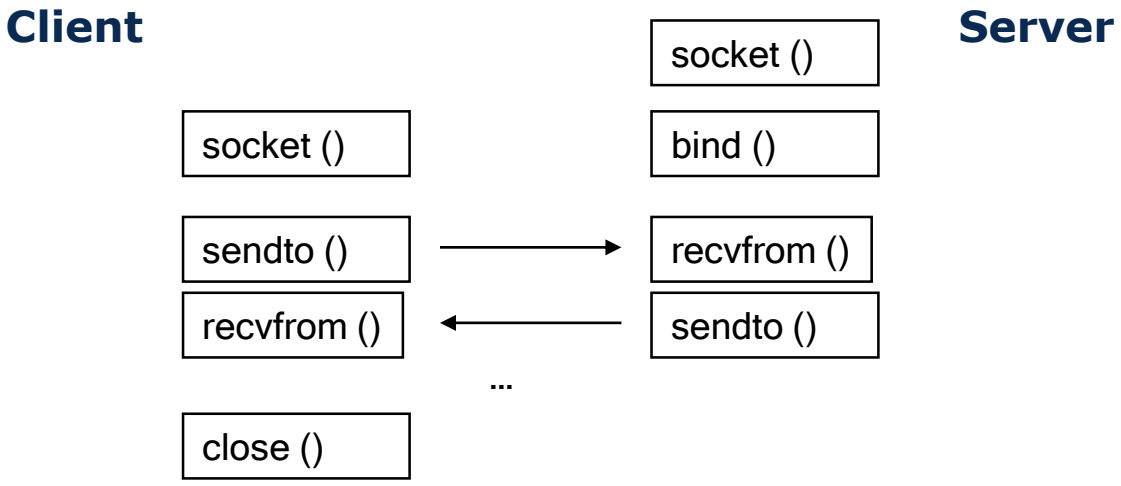
TCP Weiterentwicklungen

TCP Vegas reagiert nicht nur auf Stauprobleme,
sondern betreibt auch vorausschauende Stausteuerung.

- Bei drohendem Paketverlust wird Senderate reduziert
  (erkennbar an Vergrößerung der Wartezeiten auf Quittungen).

UDP – Nutzung (Socketprogrammierung)

Funktion	Inhalt
socket	Anforderung Kommunikationsendpunkt
bind	Festlegen Portnummer
sendto	Senden
recvfrom	Empfangen
close	Schließen Socket



TCP – Nutzung (Socketprogrammierung)

Funktion	Inhalt
socket	Anforderung Kommunikationsendpunkt
bind	Festlegen Portnummer
listen	Empfangspuffer und Warten auf Verbindungsaufnahme
accept	Annahme Verbindung
connect	Anforderung Verbindung
send	Senden
recv	Empfangen
close	Schließen Socket

zusätzlich:

„select" zum Warten auf Nachrichten an mehreren Sockets

TCP – Nutzung (Socketprogrammierung)