Charakteristische Fragestellungen der Schicht 4

Kapitel 8

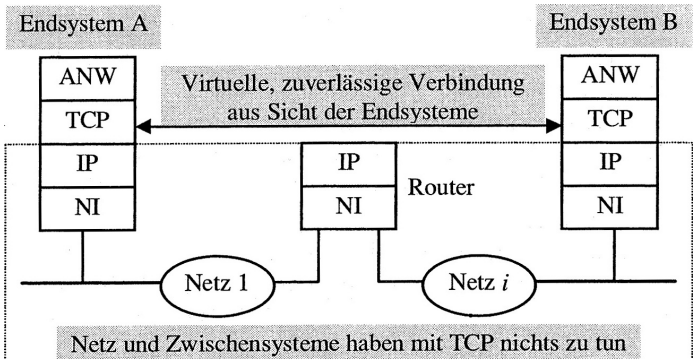
Charakteristische Fragestellungen der Schicht 4

Kapitel: 8.1:
Internet Transportprotokolle

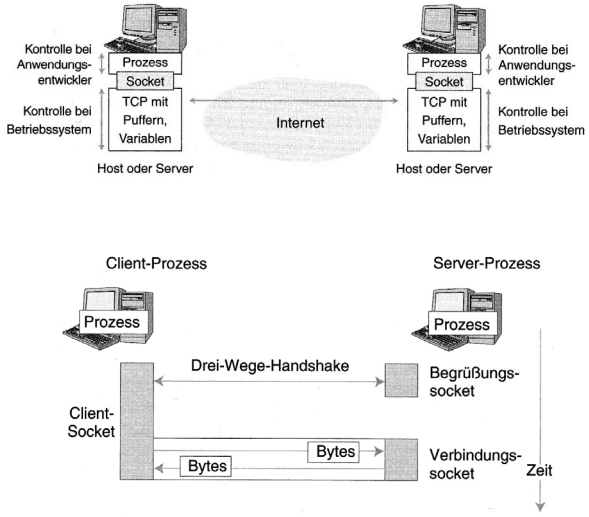
TCP (1): Überblick

- TCP (Transport Control Protocol) unterstützt E2E-Transportverbindungen (P2P), vollduplex, mit Fehlerbehandlung und Flusssteuerung (Überlastkontrolle)
- TCP ist Byte-Stromorientiert, Sequenz- und Quittungsnr. beziehen sich auf Bytes
- TCP bietet VO-Dienst, Aufbau mit 3-way-handshake
- TCP-Nutzer sind über Sockets adressierbar
  - SocketNr=(IP-Adresse Host, lokale PortNr)
  - TCP-Verbindung=(socket1, socket2)
- TCP unterstützt ein Multiplexen von Anwendungen
- TCP kann Dienstdaten zwischenspeichern, tatsächliche Transport-Blockung kann TCP-Nutzer nicht sehen

TCP (2): Einbettung TCP



TCP (3): Einbettung von TCP über Sockets (1)



Socket programming

Goal: learn how to build client/server application that communicate using sockets

Socket API

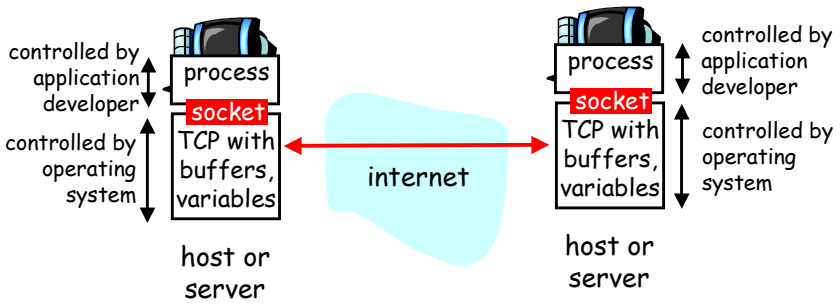
- introduced in BSD4.1 UNIX, 1981
- explicitly created, used, released by apps
- client/server paradigm
- two types of transport service via socket API:
  - unreliable datagram
  - reliable, byte stream-oriented

socket
a *host-local*, *application-created/owned*, *OS-controlled* interface (a "door") into which application process can both send and receive messages to/from another (remote or local) application process

Socket-programming using TCP

Socket: a door between application process and end-end-transport protocol (UCP or TCP)

TCP service: reliable transfer of bytes from one process to another



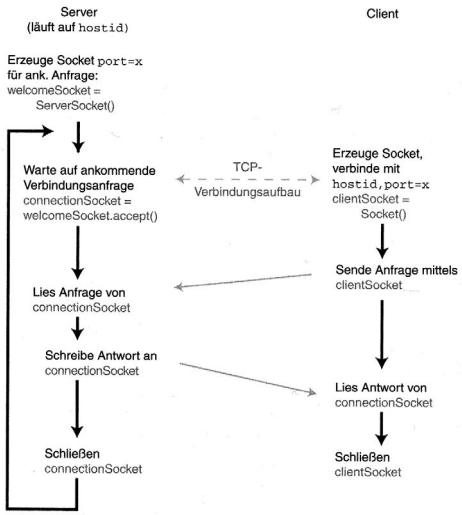
Socket programming with TCP

- Client must contact server
  - server process must first be running
  - server must have created socket (door) that welcomes client's contact
- Client contacts server by:
  - creating client-local TCP socket
  - specifying IP address, port number of server process
- When client creates socket: client TCP establishes connection to server TCP
- When contacted by client, server TCP creates new socket for server process to communicate with client
  - allows server to talk with multiple clients

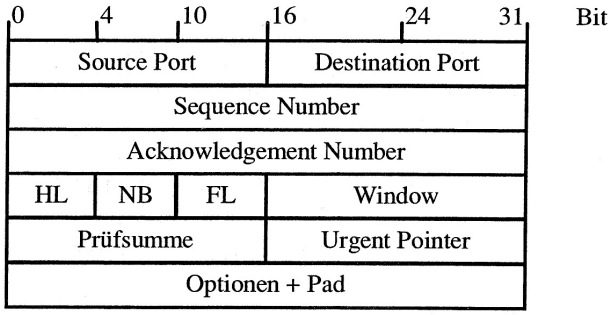
application viewpoint

TCP provides reliable, in-order transfer of bytes ("pipe") between client and server

TCP (4): Einbettung von TCP über Sockets (2)



TCP (5): Protokoll-Header



TCP (6): Protokoll-Header

- Eine TCP-PDU wird oft auch als Segment bezeichnet
- Source Port, Destination Port
  - kennzeichnen Anwendungsprozesse, z.B. wellknown ports
- Sequenznr
  - Senden und Empfangen (zählt auf Byte-Strom)
  - Sequenznr. eines Segments ist Bytestromnr. des ersten Bytes im Segment
- HL HeaderLength (4 Bit) in Anzahl 32 Bit-Wörter
- NB (nicht benutzt, 6 Bits)

TCP (7): Protokoll-Header

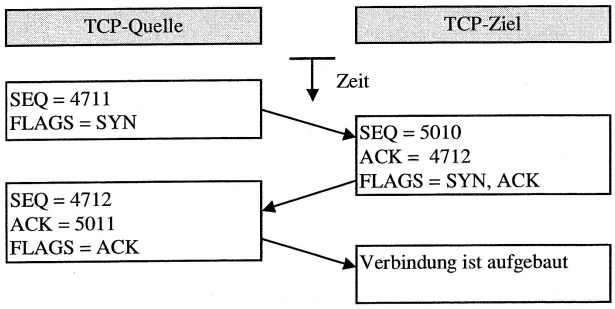
- FL (Flags, 6bits)
  - URG=1
    - Urgent Field benutzt (entspricht interrupt data)
  - ACK=1
    - Quittungssequenznr gültig
  - PSH
    - pushed data werden sofort gesendet
  - RST
    - reset connection oder Aufbauwunsch abgelehnt
  - SYN
    - Aufbauwunsch (SYN=1, ACK=0), Quittung dazu (SYN=1, ACK=1)
  - FIN
    - Abbauwunsch, keine weiteren Daten

TCP (8): Protokoll-Header

- Fenstergröße (variabel)
  - Anzahl der Bytes die ab letzter Quittung gesendet werden dürfen
- Prüfsumme
  - Einerkomplement der Summe aller 16-Bit-Worte über Pseudoheader, TCP-Header und –Rumpf. Pseudoheader enthält aus dem IP-Header die Felder Quell-/Zieladresse, Protokoll u. Länge des TCP-Segments
- Urgent Pointer
  - zeigt auf letztes Byte in einer Kette dringlicher Daten (out of band data)
- Options
  - Wählbare Eigenschaften z.B. Max SegmentSize, Timestampoption
- TCP-Ports
  - reserviert: 1-255 für TCP-Anwendungen, 256-1023 für Unix-Anwendungen
  - registriert: 1024-49151 durch IANA
  - privat, dynamisch: 49152-65535

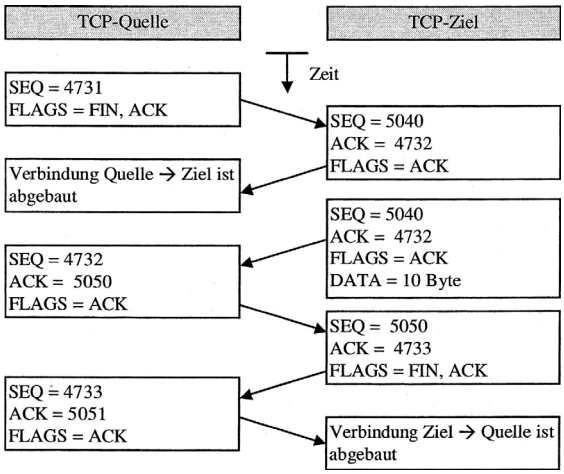
TCP (9): Verbindungsaufbau

- 3-way-handshake

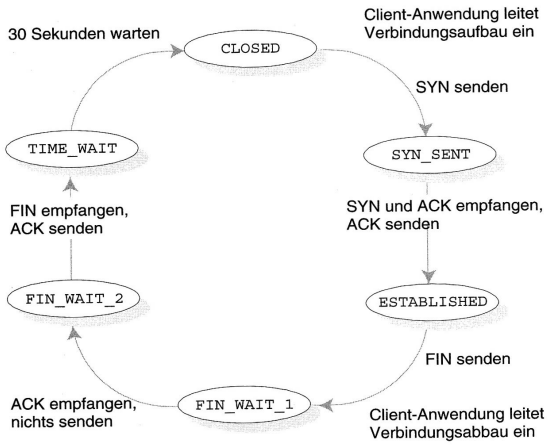


TCP (10): Verbindungsabbau

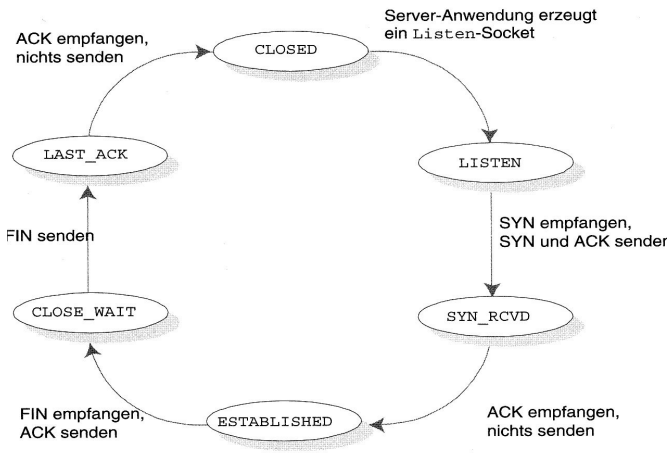
- Beidseitiger Abbau



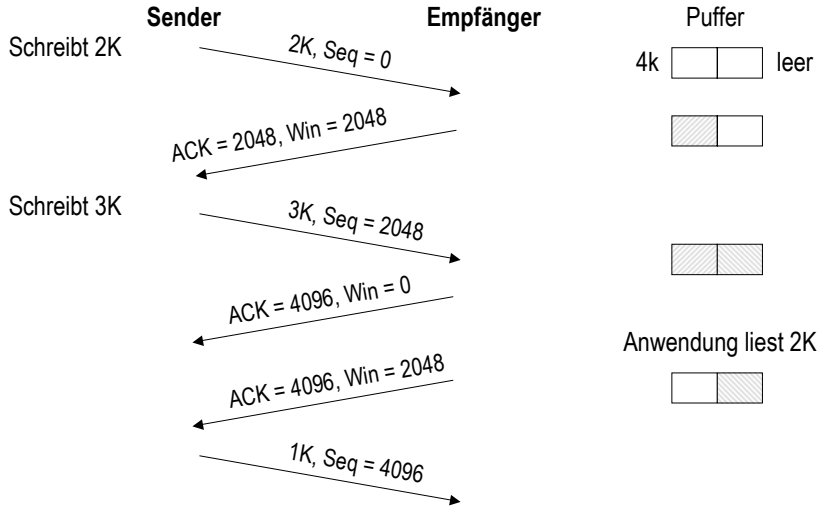
TCP (11): TCP-Zustände im Client



TCP (12): TCP-Zustände im Sever

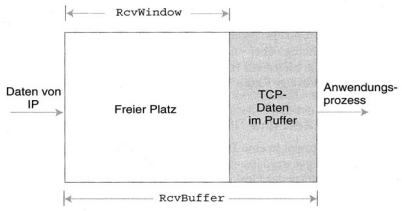


TCP (13): Fenstermanagement (1)



TCP (14): Fenstermanagement (2)

- Sender (A) hält Variable:
  - RcvWindow
  - LastByteSent
  - LastByteAcked
- Empfänger (B) hält: ▪ RcvBuffer
  - LastByteRead: gelesen vom Anwendungsprozess in B
  - LastByteRcvd: gelesen in B von Netz für Empfangspuffer
- RcvWindow = RcvBuffer - (LastByteRcvd - LastByteRead)
  LastByteSent - LastByteAcked ≤ RcvWindow



TCP (15): Timer in TCP

Timerfunktion	Bedeutung
Überwachung des Verbindungsaufbaus	Steuert die Wiederholung des Verbindungsaufbaus bis zum eventuellen Abbruch.
Retransmission Timer	Steuert die Wiederholung von Segmenten, die innerhalb der erwarteten Zeitspanne nicht bestätigt wurden.
Persist Timer	Zur periodischen Abfrage der aktuellen Fenstergröße eines nicht bereiten Empfängers.
Keepalive Timer	Überprüfung der Erreichbarkeit des entfernten Systems nach längeren Kommunikationspausen.
Quiet Timer	Stellt sicher, dass nach dem Neustart eines Endsystems dieses für die Dauer MSL (Maximum Segment Lifetime) keine TCP-Verbindung aufbaut. Damit wird eine unerwünschte Interaktion mit vielleicht noch bestehenden, alten Segmenten verhindert.
2MSL Timer	Wartet beim Verbindungsabbau das Doppelte der MSL ab, um einen möglichen Verlust des letzten ACK-Segments zu verhindern.

TCP (16): Weitere Eigenschaften

- Slow Start
  - Fenstergröße zu Anfang klein gewählt
- Congestion Avoidance
  - Timerüberläufe führen zur Reduktion der Senderate
- Nagle-Algorithmus
  - Blocking zur Vermeidung zu kurzer TCP-Segmente
- Karn-Algorithmus
  - Verbessern RTD-Schätzung und Anpassen Timer
- Flusssteuerung über Fenstermechanismus (16 Bit-Wert)
- Summenquittung über Piggy-backing
- Path MTU Discovery bestimmt kleinste Max Transmission Unit auf Pfad
- Fehlerbehandlung durch Prüfsumme und Sequenznr.

TCP (17): Überlastkontrolle (1)

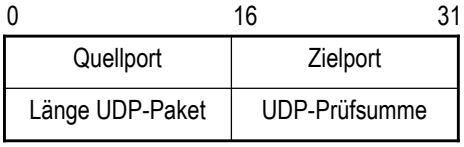
- Jede Seite unterhält Variable:
  MaximumSegmentSize MSS, Überlastfenster CongWin und Threshold
- CongWin steuert Einspeisung ins Netz:
  LastByteSent - LastByteAcked ≤ min(CongWin, RcvWin)
- Startphase: CongWin=MSS
  Wird Segment bestätigt vor dessen Timerablauf, dann Erhöhung von CongWin um 1 MSS und Senden von 2 MSS
- Nach 2 RTD: CongWin = 4 MSS, nach 3 RTD dann 8 MSS, bis Treshold erreicht, dann Ende Slow-Start-Phase und Fenster wächst alle RTD nur noch linear (Congestion Avoidance Phase), solange Quittung vor Timeout eintrifft

TCP (18): Überlastkontrolle (2)

- Bei Timeout Zurücksetzen von Threshold auf 0,5CongWin und CongWin=1 MSS
- Obige Algorithmus-Variante heißt auch Tahoe-Algor. oder AIMD-Algor. (Additive Increase, Multiplicative Decrease)
- Verbesserung im Vegas-Algor., seit 1998 auch Reno-Algor.
- TCP-Übertragungsrate (mittlerer Durchsatz):
  0,75W * MSS/RTD, wobei W max. Überlastfenster vor Verlustsituation.

UDP: User Datagram Protocol

- UDP ist ein verbindungsloses, unzuverlässiges Transportprotokoll
  (kein Verbindungsaufbau, kein Verb.-Status, unregulierte Senderate)
- Portnummer für UDP und TCP können verschieden sein
- TFTP, DNS, RPC, SNMP werden z.B. über UDP abgewickelt
- Header festgelegt in RFC 768
  (nur 8 Byte im Gegensatz zu 20 Byte bei TCP)



UDP-Header

Socket programming with UDP

UDP: no "connection" between client and server

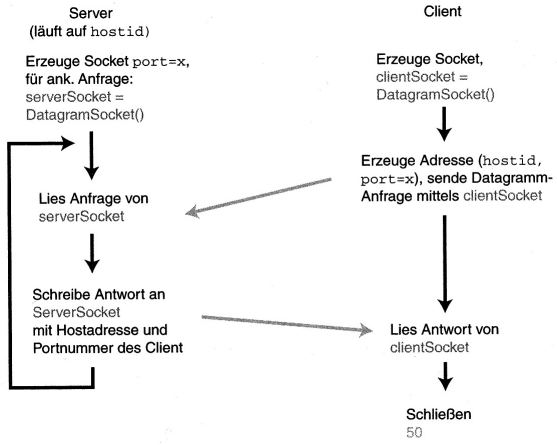
- no handshaking
- sender explicitly attaches IP address and port of destination
- server must extract IP address, port of sender from received datagram

UDP: transmitted data may be received out of order, or lost

application viewpoint

UDP provides unreliable transfer of groups of bytes ("datagrams") between client and server

Client-Server-Anwendung mit UDP



Fragen zu Kapitel 8.1 (1)

- Wann ist es u.U. sinnvoll, eine Anwendung über UDP, statt über TCP zu betreiben
- Wie sichert man einer Anwendung einen zuverlässigen Datentransfer, auch wenn sie über UDP läuft?
- *A* sendet 2 TCP-Segmente an *B*. Das erste habe die Segmentnr. 90, das zweite 110. Wie viele Daten enthält das erste Segment? Wie lang ist es insgesamt? Wenn z.B. das erste Segment verloren geht, das zweite aber bei *B* ankommt, wie lautet die Segmentnr. in der Bestätigung von *B* nach *A*?
- Welche Mechanismen bietet TCP für eine zuverlässige End-End-Verbindung?

Fragen zu Kapitel 8.1 (2)

- Welches Transportprotokoll unterstützt ein Multiplexen von Anwendungen?
- TCP erledigt Flusssteuerung über Credits statt über Fenstertechnik. Diskutieren Sie Vor- und Nachteile.
- Die Fragmentierung von Datagrammen und das Zusammensetzen wird von IP gemacht und ist für TCP unsichtbar. Heißt das, das sich TCP über die Reihenfolgesicherung von IP-Paketen keine Gedanken machen muss?