Router: Anforderungen und Trends

- Klassische Aufgaben eines Routers:
  - Matching des längsten Präfixes in der Routing-Tabelle
  - TTL-Feld dekrementieren
  - Header des Ebene 2-Protokolls bzgl. des Next-hop-Routers anfügen und ggf. Prüfsummen etc. der Ebene 2 berechnen und anfügen
  - ggf. ICMP-Fehlermeldung generieren, Statistiken führen

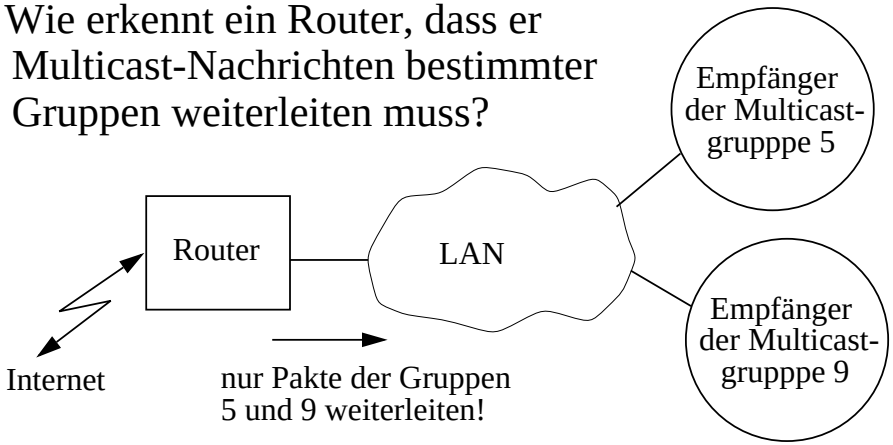
- Neue Anforderungen
  - neue Services wie Multicast, Sprache, Quality of Service, Sicherheit etc. erfordern ggf. (bei hoher Datenrate!) eine Analyse und spezielle Behandlung einzelner Datenpakete

- Router sind kritische Internet-Infrastrukturkomponenten
  - zu verarbeitendes Datenvolumen erhöht sich schnell; bei 10 Gb/s und 1000 Byte-Paketen z.B. 1 000 000 lookups pro Sekunde notwendig
  - zukünftig Terabit-Netze --> Gefahr, dass Router einen Engpass bilden
  - lookup bisher mit geeigneten Datenstrukturen / Algorithmen (z.B. Hashing oder Patricia-Bäume); neuere Verfahren sind patentiert oder nicht veröffentlicht: Minimierung der Speicherzugriffe, oft unter Verwendung sehr grosser Speicher
  - sehr leistungsfähige Backplane (z.B. crossbar statt Bus, oder shared memory für Input- und Outputports, wobei nur der header bewegt wird)
  - Verwendung schneller Puffer und Caches
  - modularer Aufbau; “line cards” für diverse Protokolle und Medien
  - zukünftig vermehrter Einsatz von Parallelität (bis auf die Bitebene mittels bit slicing und data striping)

IP-Multicasting

- Einzelnes IP-Datagramm wird an mehr als eine Station adressiert
  - die ersten vier Bits des Adressfelds im Header sind 1110
  - danach folgt eine 28-Bit-lange Multicast-Gruppennummer

- Wie erkennt ein Router, dass er Multicast-Nachrichten bestimmter Gruppen weiterleiten muss?



- IGMP (Internet Group Management Protocol)
  - spezielle IP-Pakete mit Wert = 2 im Protocol-Feld des IP-Headers
  - Router sendet periodisch Gruppenzugehörigkeitsanfragen an alle Rechner des LAN (per Broadcast)
  - durch TTL = 1 wird die Anfrage nur im LAN verbreitet
  - Rechner antworten nach einer zufälligen Zeit per LAN-Broadcast
  - andere Rechner dieser Gruppe im LAN, die das empfangen, antworten dann nicht mehr
  - Router erhält alle Antworten und aktualisiert seine Routingtabelle; mehrfach nicht bestätigte Gruppen werden gelöscht
  - tritt ein Rechner einer Gruppe bei, so sendet er sofort eine entsprechende Mitteilung an alle Router im LAN

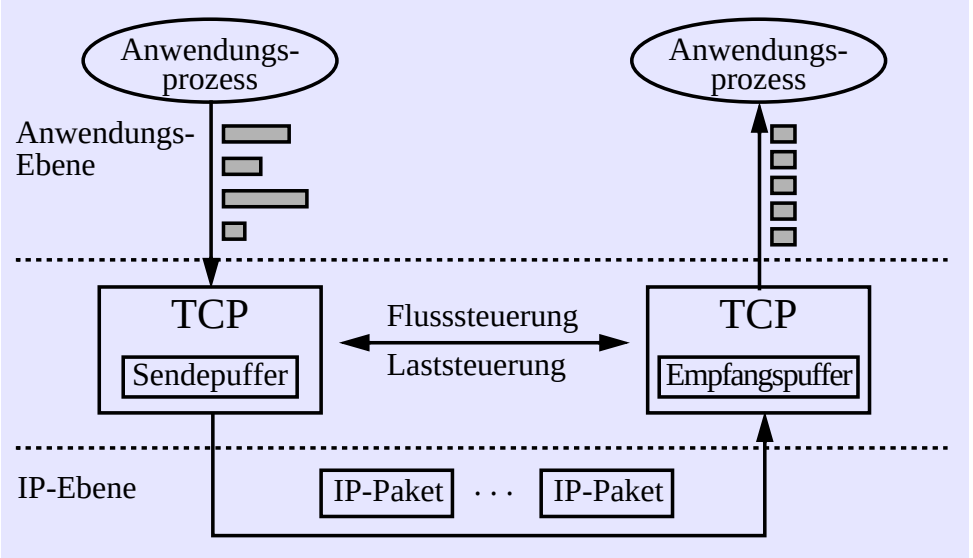
TCP (Transmission Control Protocol)

- verbindungsorientiert (Ebene 4); setzt auf IP auf
- Motivation: IP hat einige “Anwendungsschwächen”:
  - Verlust von Paketen
  - ggf. Umordnung von Paketen (parallele Übertragungswege!)
  - beschränkte Länge von Paketen
- Fast jede Anwendung muss daher gewisse (i.w. die gleichen!) Massnahmen treffen
- TCP erledigt dies weitgehend für “alle”
  - i.w. mit sliding window
  - gute Parameterwahl (timeouts, Fenstergrössen) nicht einfach, da das Protokoll für sehr unterschiedliche Situationen effizient sein soll (LAN / WAN; 10kb/s - 1Gb/s; kleine / grosse Zeitschwankungen...)

- TCP bietet einen zuverlässigen Kanal für einen Bytestrom
  - zwischen zwei Ports (=Kommunikationsendpunkte)
  - voll duplex
  - unstrukturiert (nur Bytes; Nachrichtengrenzen gehen verloren), z.B.:

    send (170 Bytes); send (230 Bytes);
    receive (20 Bytes); receive (270 Bytes); receive (110 Bytes);
  - vgl. dies mit Schreiben / späteres Lesen von Bytes auf eine Platte
- Verbindungsaufbau vor Datenübertragung notwendig
  - Verbindungsabbau danach

Bytestrom



- *Flusssteuerung*: Sender soll *Empfänger* nicht überlasten
- *Laststeuerung* (congestion control): Sender soll das *Netz* nicht verstopfen
- Datenstrom wird von TCP in *IP-Pakete* aufgeteilt
  - nach dem IP-Header folgt vor den Daten ein TCP-Header
- Ein einzelnes Datenbyte verursacht so ggf. einen Overhead von einigen zig Bytes (z.B. Echo bei einem Editor)!
  - Gegenmassnahme: z.B. kurze (!) Verzögerung beim Sender, um ggf. weitere Bytes der Anwendung in das gleiche IP-Paket zu packen

Die Socket-Programmierschnittstelle

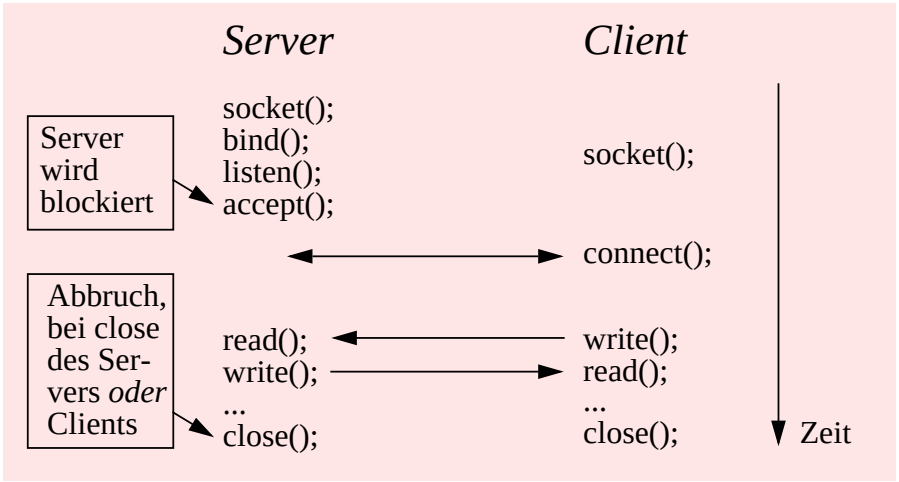
- Zu TCP (bzw. UDP) gibt es keine festgelegten “APIs”
- Bei UNIX sind dafür “sockets” als Zugangspunkte zum Transportsystem entstanden
  - etwas modernere Alternative: TLI (Transport Layer Interface)
  - sockets in leicht veränderter Form auch unter MS-Windows
- Semantik eines sockets: analog zu Datei-Ein/Ausgabe
  - ist insbesondere bidirektional (“schreiben” und “lesen”)
  - ein socket kann aber auch mit mehreren Prozessen verbunden sein
- Programmiersprachliche Einbindung (z.B. in C)
  - sockets werden wie Variablen behandelt (können Namen bekommen)
  - Beispiel in C (Erzeugen eines sockets):

```
int s;
s = socket(int PF_INET, int SOCK_STREAM, 0);
```

“Family”: Internet oder nur lokale Domäne

“Type”:Angabe, ob TCP verwendet (“stream”); oder UDP (“datagram”)

Client-Server mit Sockets (Prinzip)



- Voraussetzung: Client “kennt” die IP-Adresse des Servers sowie die Portnummer (des Dienstes)
  - muss beim connect angegeben werden
- Mit “listen” richtet der Server eine Warteschlange für Client-connect-Anforderungen ein
  - Auszug aus der Beschreibung: *“If a connection request arrives with the queue full, tcp will retry the connection. If the backlog is not cleared by the time the tcp times out, the connect will fail”*
- Accept / connect implementieren ein “Rendezvous”
  - mittels des 3-fach-Handshake von TCP
  - bei “connect” muss der Server bereits listen / accept ausgeführt haber
- Rückgabewerte bei read bzw. write: Anzahl der tatsächlich gesendeten / empfangenen Bytes

Sockets unter Java

- Paket java.net.* enthält u.a. die Klasse “Socket”
- Stream- (verbindungsorientiert) bzw. Datagrammsockets
- Beispiel (für einen Client):

```
DataInputStream in;
PrintStream out;
Socket server;
...
server = new Socket(getCodeBase().getHost(),7);
  // Klasse Socket besitzt Methoden
  // getInputStream bzw. getOutputStream, hier
  // Konversion zu DataInputStream / PrintStream:
in = new DataInputStream(server.getInputStream());
out = new PrintStream(server.getOutputStream());
...
  // Etwas an den Echo-Server senden:
out.println(...)
...
  // Vom Echo-Server empfangen; vielleicht
  // am besten in einem anderen Thread:
String line;
while((line = in.readLine()) != null)
  // line ausgeben
...
server.close;
```

Herstellen einer Verbindung

Hostname

Echo-Port

Port Nummer 7 sendet alles zurück

- Zusätzlich: Fehlerbedingungen mit Exceptions behandeln (“try”; “catch”)
  - z.B. “UnknownHostException” beim Gründen eines Socket

Bemerkung zum Echo-Port:

1) Beliebt bei Hackern: denial of service-Attacke durch Endlosschleife (gefälschter Absender127.0.0.1 als “lokaler Rechner” im IP-Paket)
2) Übung: mit Port 7 ein “ping”-Programm realisieren, das die RTT misst

Port-Nummern im Internet

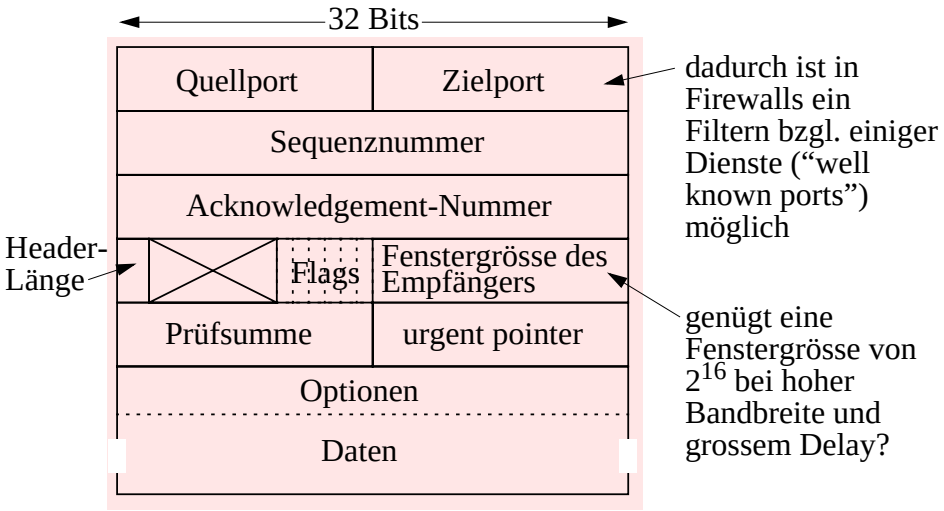
- Ports sind logische Kommunikationsendpunkte; mit einem Port ist ein Programm (“Service”) verbunden
- Der Sender (“Client”) muss den “richtigen” Port kennen, den er adressieren soll
- Ports sind numeriert
  - einige (0 - 1023) sind weltweit eindeutig (“well known port”)
  - andere werden dynamisch alloziert (und müssen den Kommunikationspartnern zuvor mitgeteilt werden)
- Die Nummern der “well known ports” werden von der IANA (Internet Assigned Numbers Authority) verwaltet

> The IANA is chartered to act as the clearinghouse to assign and coordinate the use of numerous Internet protocol parameters.
>
> The common use of the Internet protocols by the Internet community requires that the particular values used in these parameter fields be assigned uniquely. It is the task of the IANA to make those unique assignments as requested and to maintain a registry of the currently assigned values.

echo	7	Echo
daytime	13	Daytime
ftp-data	20	File Transfer[Default Data]
ftp	21	File Transfer[Control]
telnet	23	Telnet
smtp	25	Simple Mail Transfer
domain	53	Domain Name Server
finger	79	Finger
http	80	World Wide Web HTTP
kerberos	88	Kerberos
pop3	110	Post Office Protocol - Version 3
nntp	119	Network News Transfer Protocol
ntp	123	Network Time Protocol
bgp	179	Border Gateway Protocol
irc	194	Internet Relay Chat Protocol
imap3	220	Interactive Mail Access Protocol

Auszug aus der Liste der well-known ports

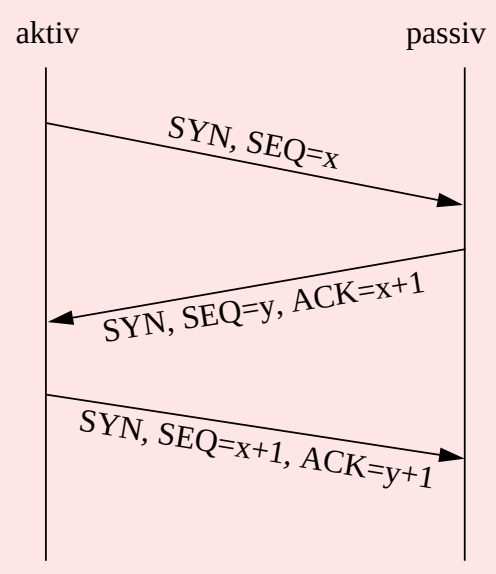
Der TCP-Header



- Vorher (im IP-Header) stehen die IP-Adressen
- Jedes Byte wird numeriert (für sliding window)
  - Sequenznummer bzw. Ack.-Nummer
  - wrap-around der Sequenznummern bei 100 Mb/s nach 6 min möglich!
- Ack.-Nr. und Fenstergrösse dienen der Rückmeldung vom Empfänger zum Sender (ggf. huckepack mit Nutzdaten in Rückrichtung versandt)
- Mit den Flags kann folgendes ausgedrückt werden:
  1) URG: “urgent”: dringende Daten (z.B. cntrl-C)
  2) ACK: Acknowledgement-Nummer ist gültig
  3) PSH: Empfänger soll nicht puffern (z.B. <CR> bei Zeilenende)
  4) RST: Reset der Verbindung (nach einem erkannten Problem)
  5) SYN: Aufbau einer Verbindung (Synchronisieren erwarteter Sequenznummern)
  6) FIN: Beenden einer Verbindung

Verbindungsauf- und abbau

- Aufbau der Duplex-Verbindung durch 3-fach-Handshake



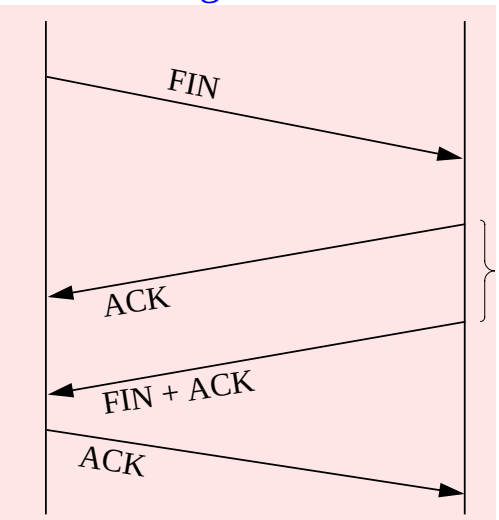
Austausch und Bestätigung initialer Sequenznummern, um alte von neuen Daten auch nach einem Rechnerabsturz zu unterscheiden

Hier aktiver Partner (“Client”) und passiver Partner (“Server”)

Gleichzeitiger Verbindungsaufbau von zwei aktiven ist auch möglich (resultiert aber in einer einzigen Verbindung)

Denkübung: Was kann geschehen, wenn Kontrollnachrichten verloren gehen oder nach langer Zeit “aus dem Nichts” auftauchen?

- Verbindungsabbau durch vier Kontrollnachrichten

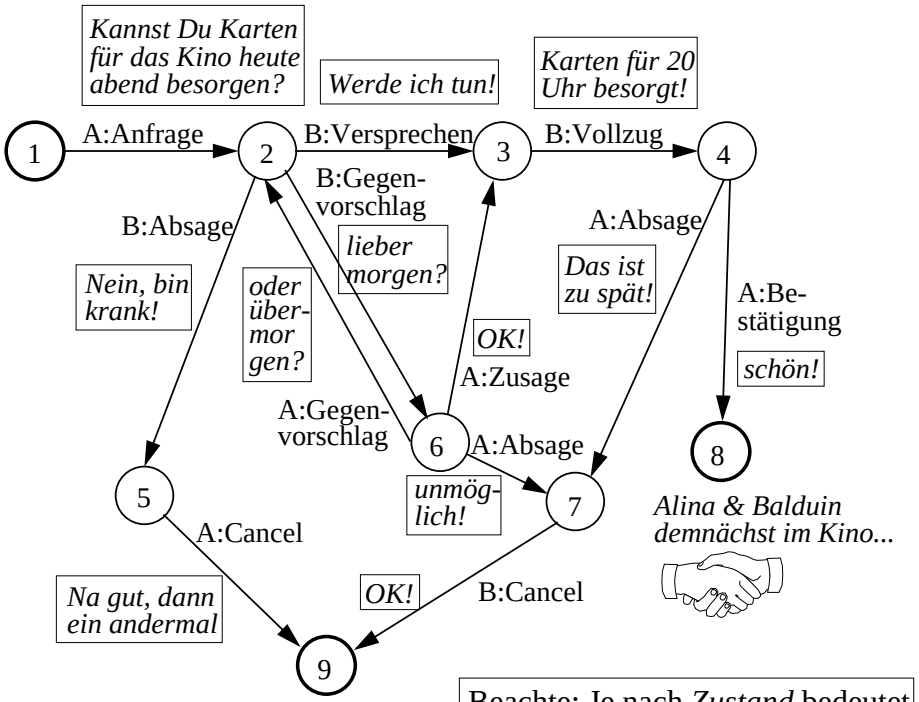


Denkübung: Wie kann sich ein Kommunikationspartner sicher sein, dass der andere die Verbindung ebenfalls abbaut, wenn Kontrollnachrichten (z.B. ACK) verloren gehen können?

Hier können noch Daten gesendet gesendet werden

Es gibt auch den Verbindungsabbruch (statt Abbau); dann können ggf. gesendete Daten verloren gehen

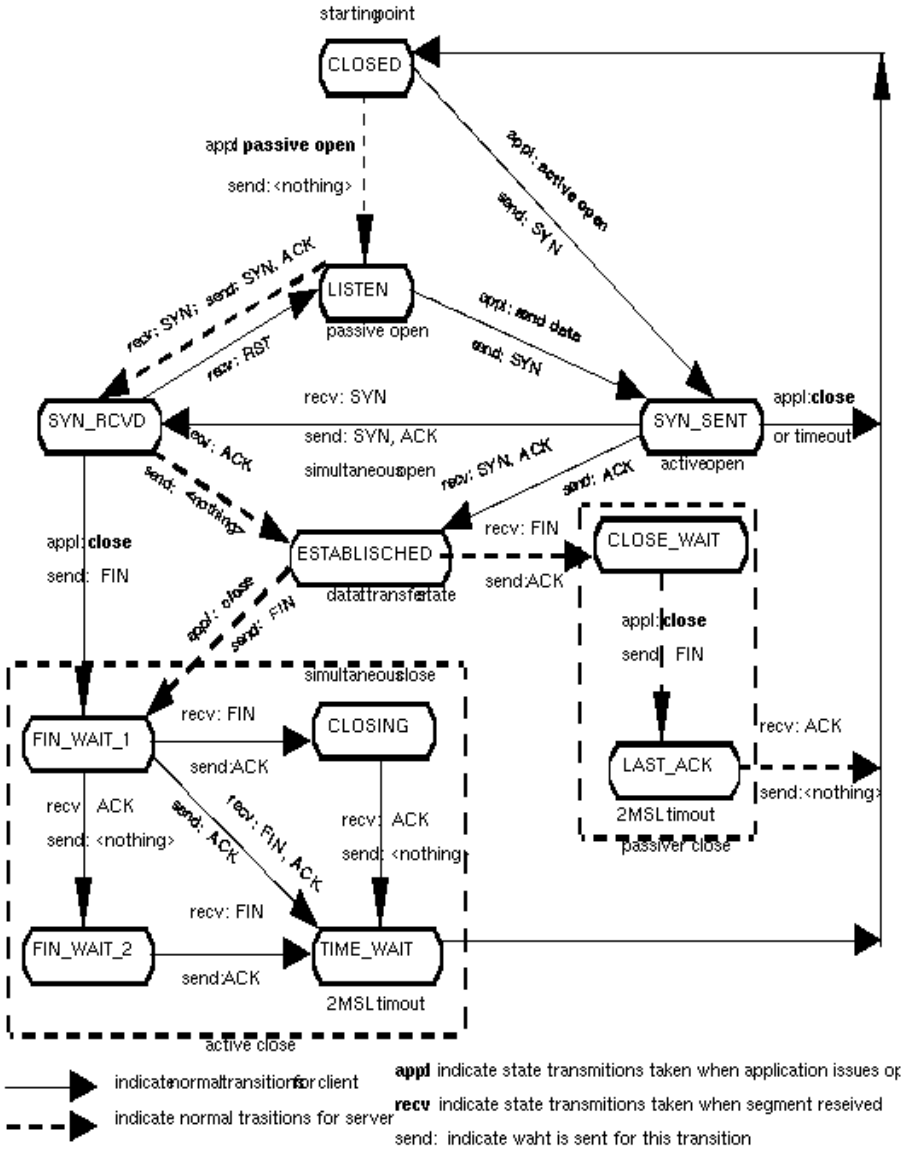
Zustandsübergangsdiagramme für Protokolle



- "Konversationszustände" --> *endl. determ. Automat*
- Ggf. *Missverständnisse* bei Nachrichtenverlust!
- Koordination von Handlungen --> *Protokolle*
- vgl. *Sprechakttheorie*: Allgemeines Schema für viele möglichen "Sprechaktfolgen" (z.B. 1,2,6,2,3,4,8)

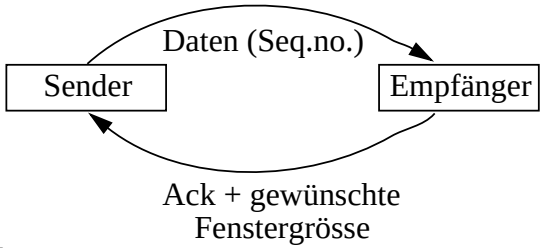
TCP-Zustandsdiagramm

Vgl. dazu die Literatur (z.B. Tanenbaum p. 532; Fig. 6-28)



TCP: Fluss- und Laststeuerung

- Sliding window-Protokoll



- Fenstergrösse adaptiv
  - es gibt zwei Fenstergrössen:
    1) Wunsch des Empfängers
    2) Grösse des "congestion window", das durch TCP selbst aufgrund der Beobachtung der Netzlast dynamisch verändert wird
  - Sender richtet sich nach dem Minimum der beiden Werte
  - Maximum ist i.a. auch durch die Puffergrösse beim Einrichten eines sockets bestimmt
  - in TCP-Paketen nur 16 Bits für die Fenstergrösse vorgesehen --> Skalierungsfaktor vor Start der Kommunikation "aushandeln"
  - ideal wäre das Bandbreite-Delay-Produkt (wieso?)

z.B. mittels ping ermittelbar

Denkübung: einfache Zeit oder round-trip-Zeit?

- Timeouts für Retransmissionen adaptiv
  - richtigen Wert zu finden ist eine Kunst!
- Bestätigungen durch Acks sind kummulativ
  - kummulative Acks bei Hochgeschwindigkeit allerdings problematisch, selektives Ack / retransmit wäre effizienter

Congestion Window bei TCP

- Problem der Aufschaukelung von Netzüberlastungen:
  Netzüberlastung --> Timeouts --> Paketwiederholungen --> noch mehr Last
- TCP ist an sich "selbsttaktend" durch die Verwendung von Acknowledgements bei sliding window
- Um einer Netzüberlastung gegenzusteuern, wird bei Überlast das congestion window verkleinert
- Indikatoren für eine Netzüberlastung:
  - timeout bzgl. eines erwarteten Ack
  - Empfang einer "source quench"-ICMP-Nachricht

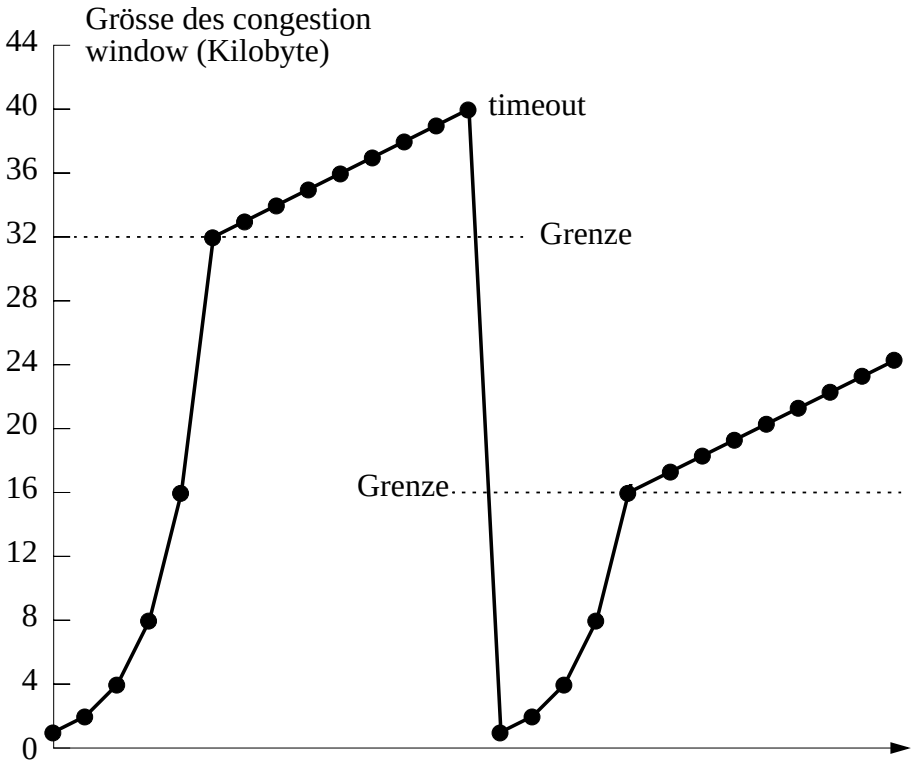
- Es gibt verschiedene Strategien zur dynamischen Veränderung des congestion window, u.a.:

1) "additive increase / multipicative decrease"
   - bei jedem timeout die Fenstergrösse halbieren
   - bei Erfolg (Ack bzgl. eines TCP-Segments kommt vor Ablauf des timeouts an), Fenstergrösse um ein festes Inkrement erhöhen

2) "slow start"
   - bei Erfolg Fenstergrösse verdoppeln
   - jedoch nur bis zu einem Grenzwert, ab dann linear vergrössern
   - nach einem timeout den Grenzwert auf die Hälfte der ggw. Fenstergrösse setzen und aktuelle Fenstergrösse auf einen kleinen Wert setzen

slow

Slow start



Timeout-Bestimmung bei TCP

- Problem: Varianz der Übertragungszeiten ist sehr gross

- Man arbeitet mit einem Schätzwert RTT für die round-trip-Zeit, der laufend (gleitend) angeglichen wird:

$$RTT = \alpha\, RTT + (1-\alpha)\, M$$

wobei M die gemessene round-trip-Zeit bzgl. des letzten Acknowledgements ist und α typw. auf 7/8 gesetzt wird

- Der timeout-Wert wurde dann ursprünglich so bestimmt:

$$timeout = 2\, RTT$$

- Spätere TCP-Implementierungen verwenden stattdessen:

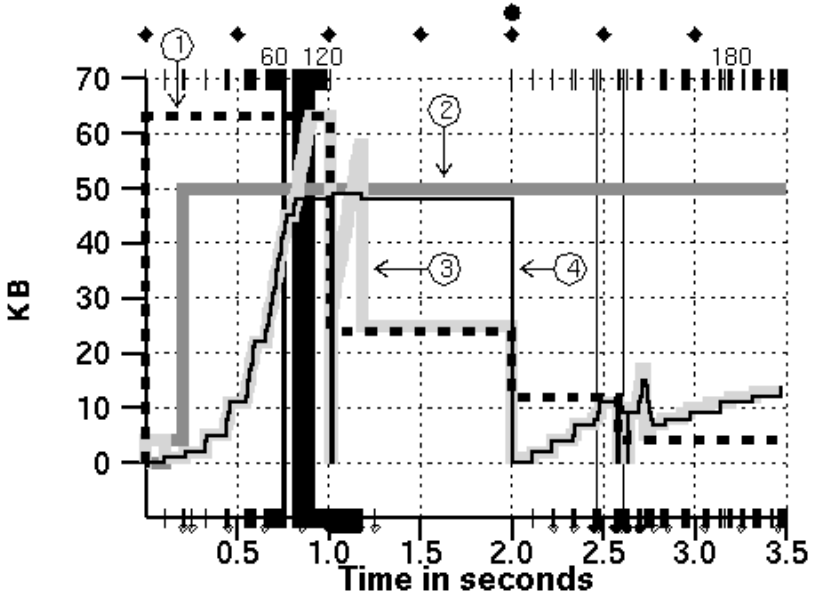
$$timeout = RTT + V$$

wobei V ein Wert ist, der aus der beobachteten Varianz der letzten round-trip-Messungen hervorgeht (Motivation: bei grösserer Varianz sollte der timeout grösser sein)

- Denkübung: Welche negativen Konsequenzen hat eigentlich ein zu grosser / zu kleiner timeout-Wert?

TCP-Benchmarks

1) Markierung auf der x-Achse: Ack wird empfangen
2) Markierungsstrich am oberen Bildrand: IP-Paket wird versendet
3) Zahl *n* am oberen Bildrand: Wann das *n*-te Kilobyte versendet wird
4) Rauten am oberen Bildrand: Nur alle 0.5 Sekunden wird überprüft, ob ein timeout abgelaufen ist
5) Punkt am oberen Bildrand: Retransmission wegen abgelaufenem timeout

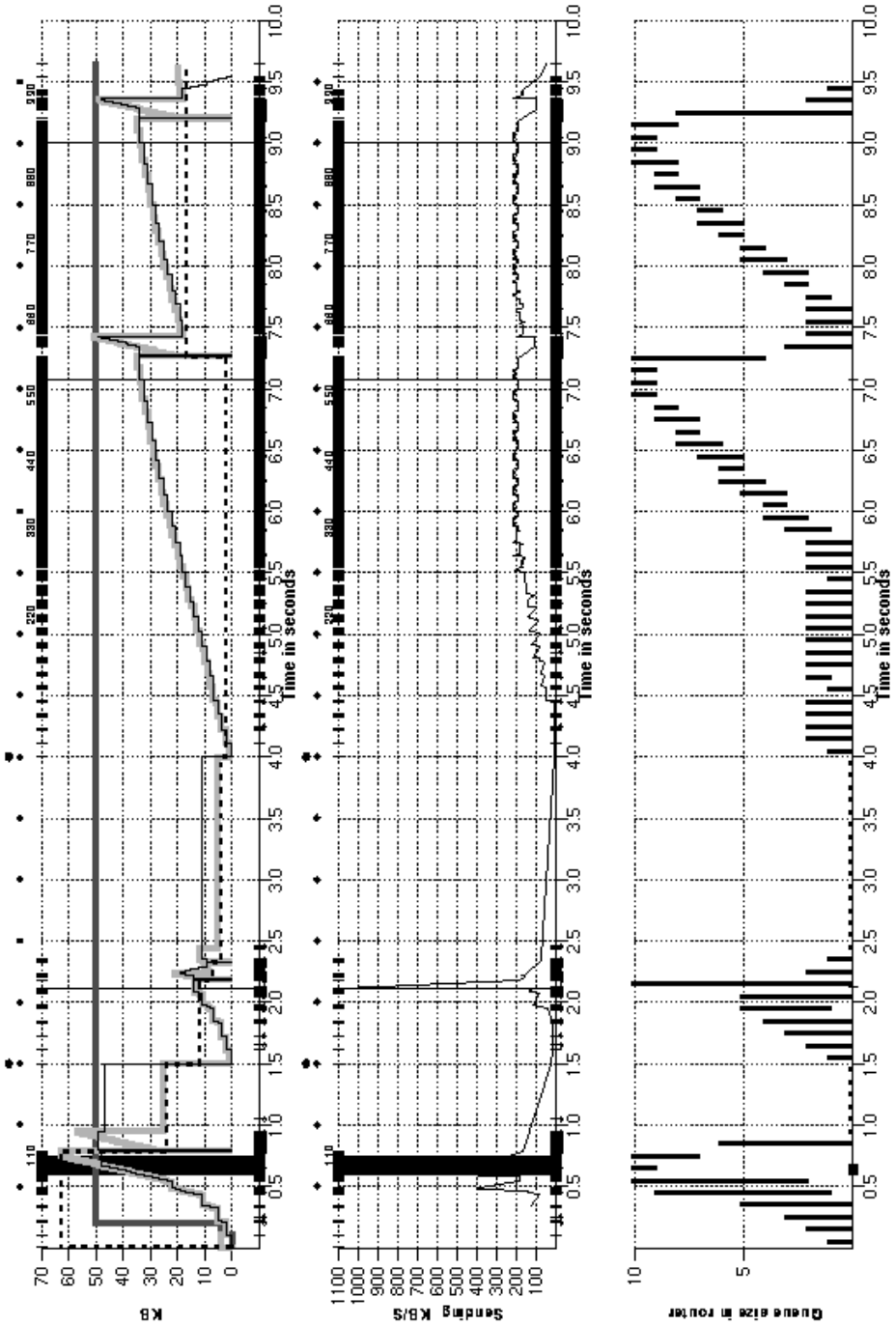


(1) gestrichelte Linie: Grenzwert exponentielle / lineare Fenstervergrösserung
(2) dunkelgraue Linie: Grösse des Sendefensters
(3) hellgraue Linie: Grösse des congestion windows
(4) dünne Linie: Anzahl der gesendeten, aber noch nicht bestätigten Bytes

---

Nächste Folie (oberes Bild) zeigt dies über einen längeren Zeitraum; im mittleren Bild ist der effektive Durchsatz angegeben; im unteren Bild die Länge einer Router-Warteschlange (max. 10). Zwischen 5.5 und 7 wird das congestion window vergrössert, der Durchsatz (mittleres Bild) bleibt jedoch gleich (offenbar ist die max. Netzbandbreite erreicht!). Die höhere "Sendeleistung" muss von den Puffern des Routers abgefedert werden!

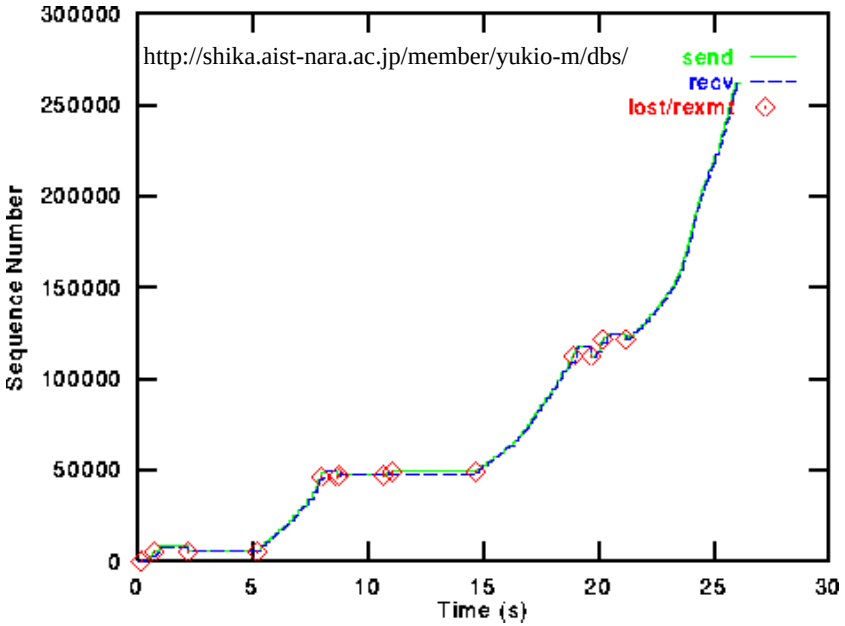
Quelle: http://excalibur.usc.edu/research/vegas/doc/vegas.html



TCP-Benchmark (2)

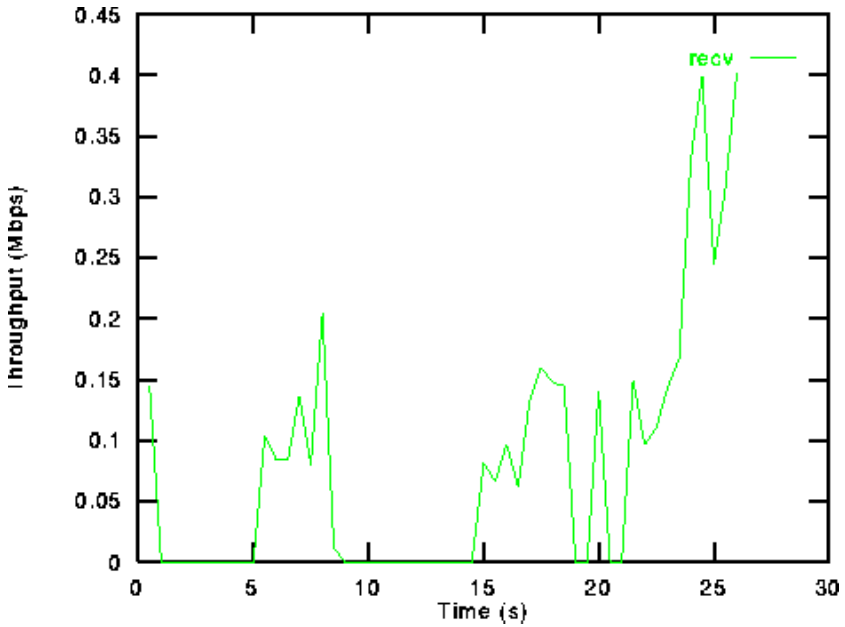
- Terrestrische Verbindung mit 1.5 Mb/s Bandbreite
  - Weitverkehrsverbindung über mehrere 100 km (typ. für Internet)
  - Beobachtung: TCP wiederholt ca. 1% aller TCP-Segmente
  - max. Fenstergrösse 8196 Byte
  - Pakete von 1024 Byte
  - typisches Szenario: Dateitransfer

- *Sequenznummer* der Bytes über die Zeit gemessen:

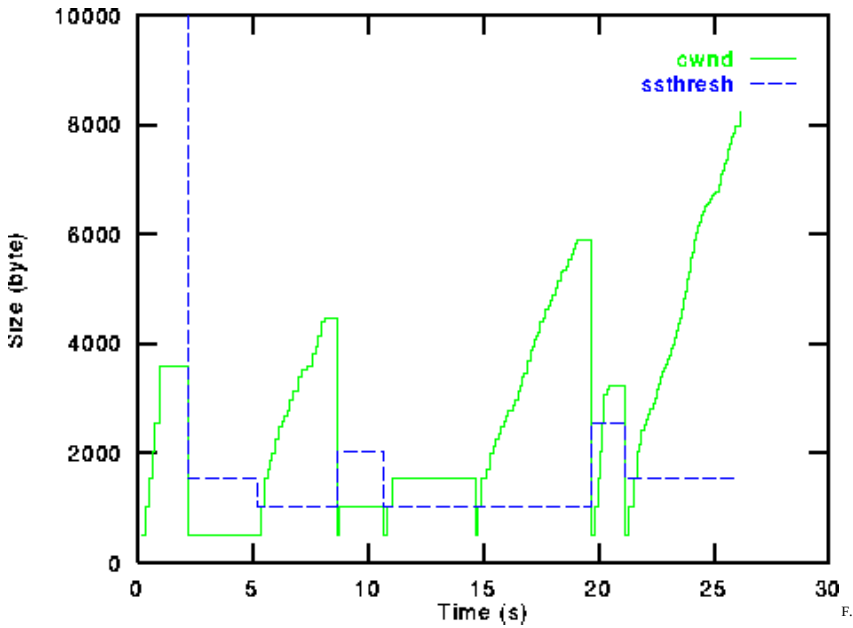


  - Anstieg ist nicht linear --> keine gleichmässige “Geschwindigkeit”
  - lange Phasen, wo nichts geschieht (“stalled”)
  - sind Retransmissionen Folge oder Ursache für die Probleme?

- *Durchsatz* im Mittel nur ca. 80 kb/s --> 5% Effizienz:

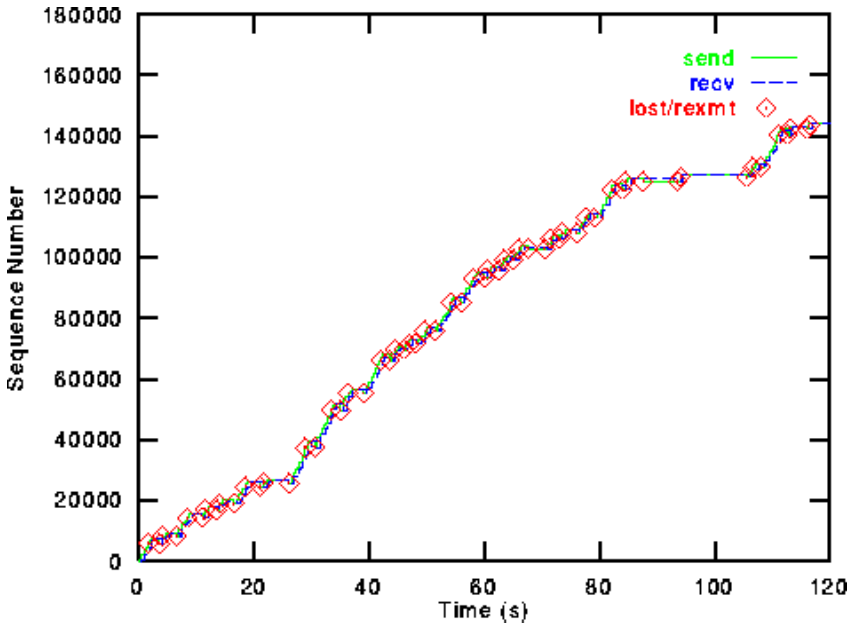


- *Congestion Window* fällt oft auf einen kleinen Wert:

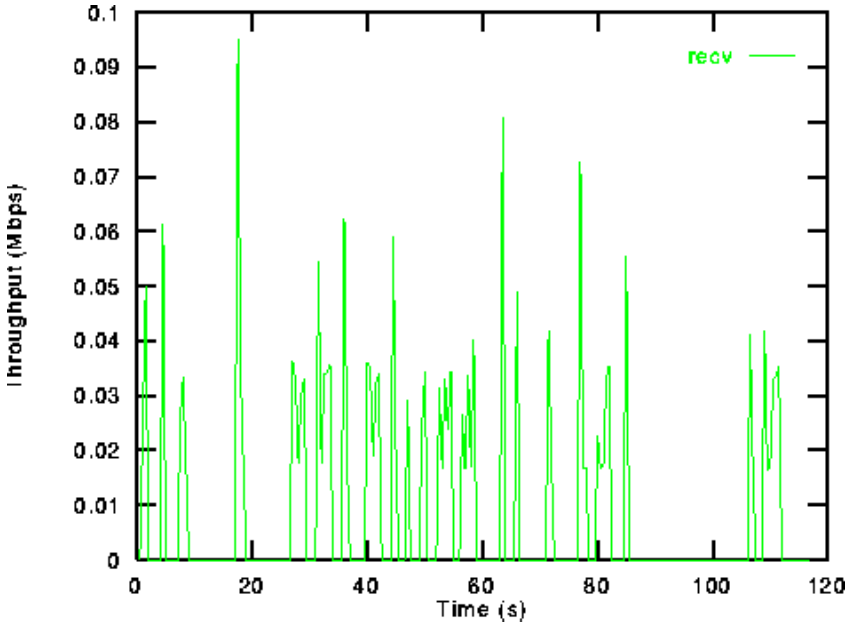


TCP-Benchmark (3)

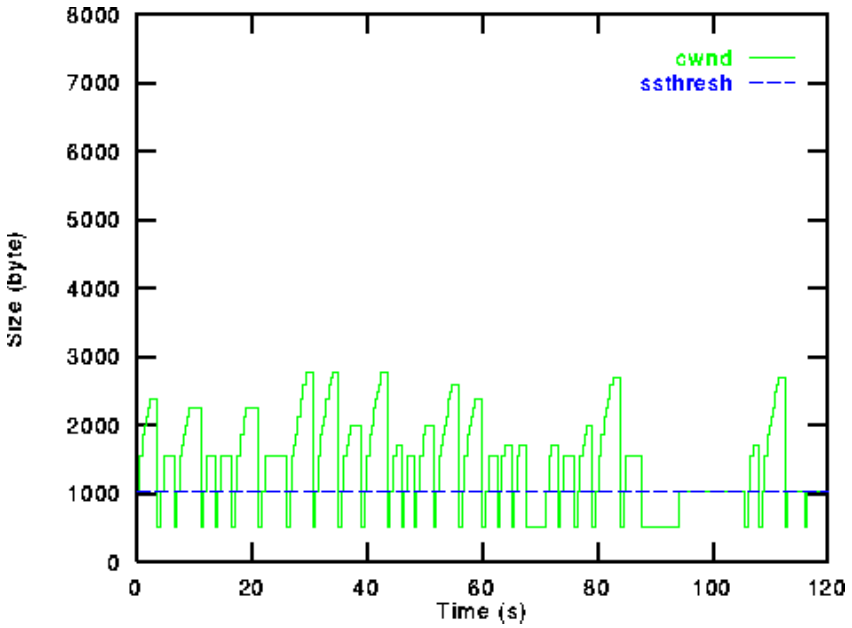
- Analoge Situation, jetzt kommt es aber zu Wiederholungen von 30% aller TCP-Datensegmente
  - z.B. aufgrund von typischen Überlastsituationen im Internet
  - nach 120 Sekunden sind diesmal erst ca. 140000 Byte übertragen



- Durchsatz nur noch ca. 10 kb/s; stark schwanken:

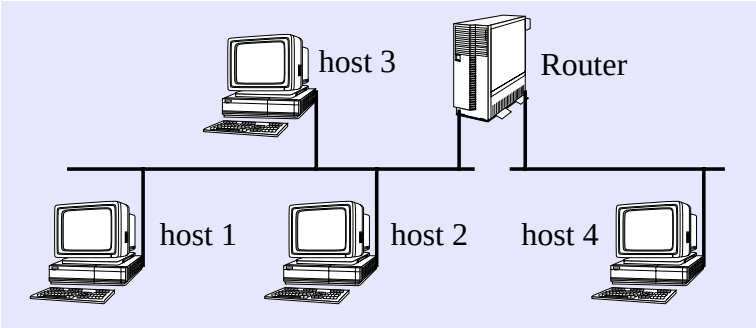


- Congestion Window kann kaum wachsen:

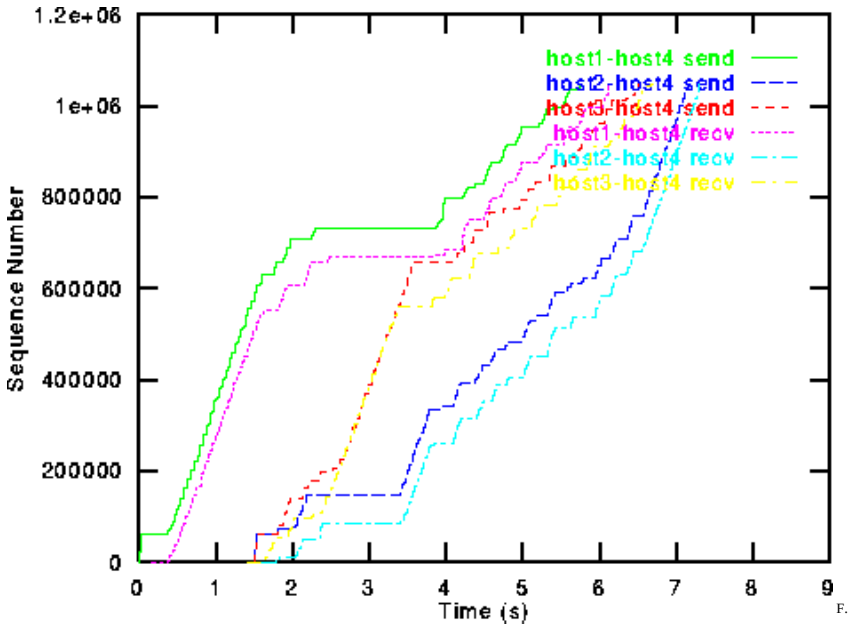


TCP-Benchmark (4)

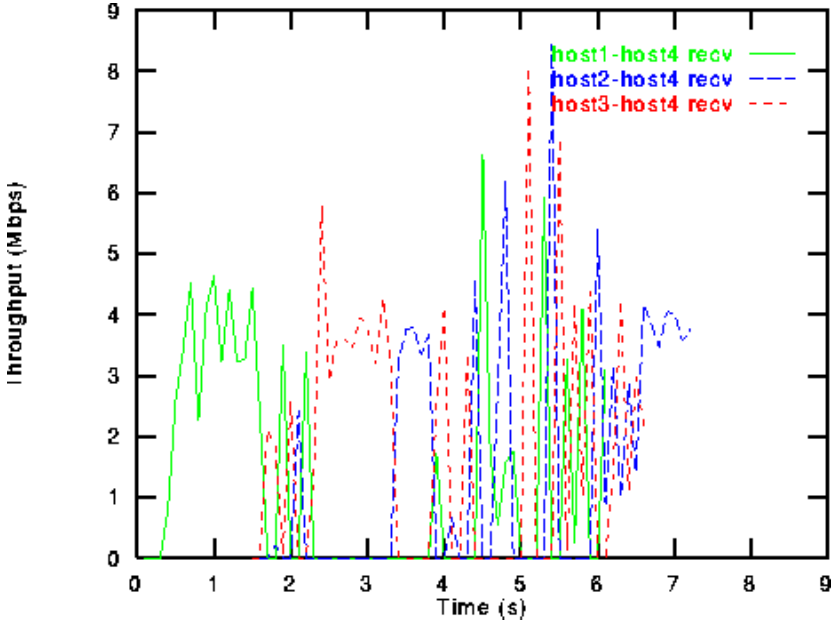
- 2 Ethernet-Segmente verbunden durch einen Router
  - LAN aus klassischem Ethernet mit 10 Mb/s



- Host 1, host 2 und host 3 senden jeweils 1000 Datenpakete zu 1024 Byte an host 4
  - host 2 und host 3 starten den Datentransfer zeitversetzt um 1 Sekunde



- Durchsatz:



- Beobachtungen:
  - Durchsatz ist selten höher als 5 Mb/s; im Mittel ca. 3.7 Mb/s
  - zeitweise Monopolisierung (z.B. von 2.5 bis 3.5 durch host 3)

- Denkübung: wie könnte man die Phänomene erklären?
  - hier spielt u.a. das Ethernet-Protokoll (CSMA/CD) und das TCP-Protokoll hinein!