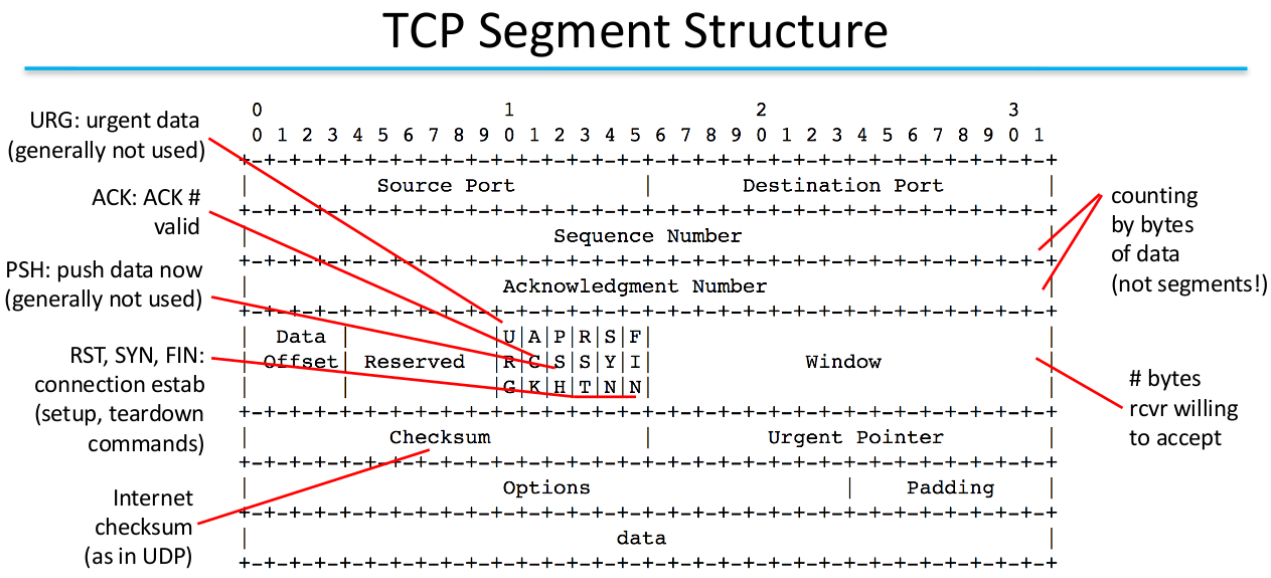
TCP: Overview

– Full

- Full duplex data
  - Bi-directional data flow in same connection
  - MSS: maximum segment size
- Connection-oriented
  - Handshaking (exchange of control messages) initializes sender, receiver state before data exchange
- Flow controlled
  - Sender will not overwhelm receiver
- Point-to-point
  - One sender, one receiver
- Reliable, in-order byte steam
  - No "message boundaries"
- Pipelined
  - TCP congestion and flow control set window size

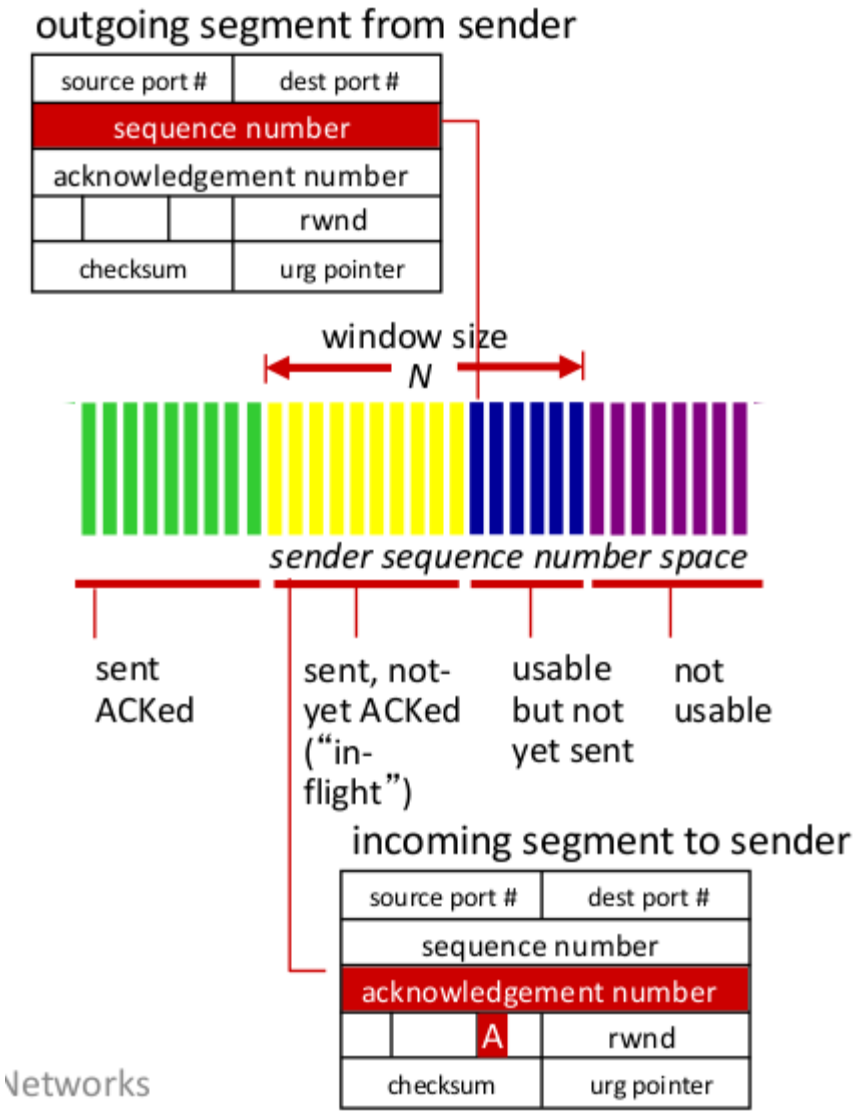
Duplex: Datenaustausch von beiden Geräten
– MSS: Maximum Segment Size – Was passsiert in darunterliegenden Schichten
→ Definiert “Maximum Transfer Unit” zwecks Datentransfer großer Medien, basierend auf darunterliegender physischen Grenzen [RFC879, RFC2460]
– Verbindungsorientert: Verbindungsaufbau vor Datenaustausch
– Flow Controlled: Datentransfer wird reguliert, sodass Empfänger verarbeiten kann.
– Reliability & In-Order-Byte Stream: Ohne Datenverlust
– Punkt2Punkt-Verbindung: Fixe Verbindung zweier Geräte :: Multipath-TCP soll dies erweitern.
– Pipeline: Multiple Datenübertragung: Selective Repeat, Go-Back-N



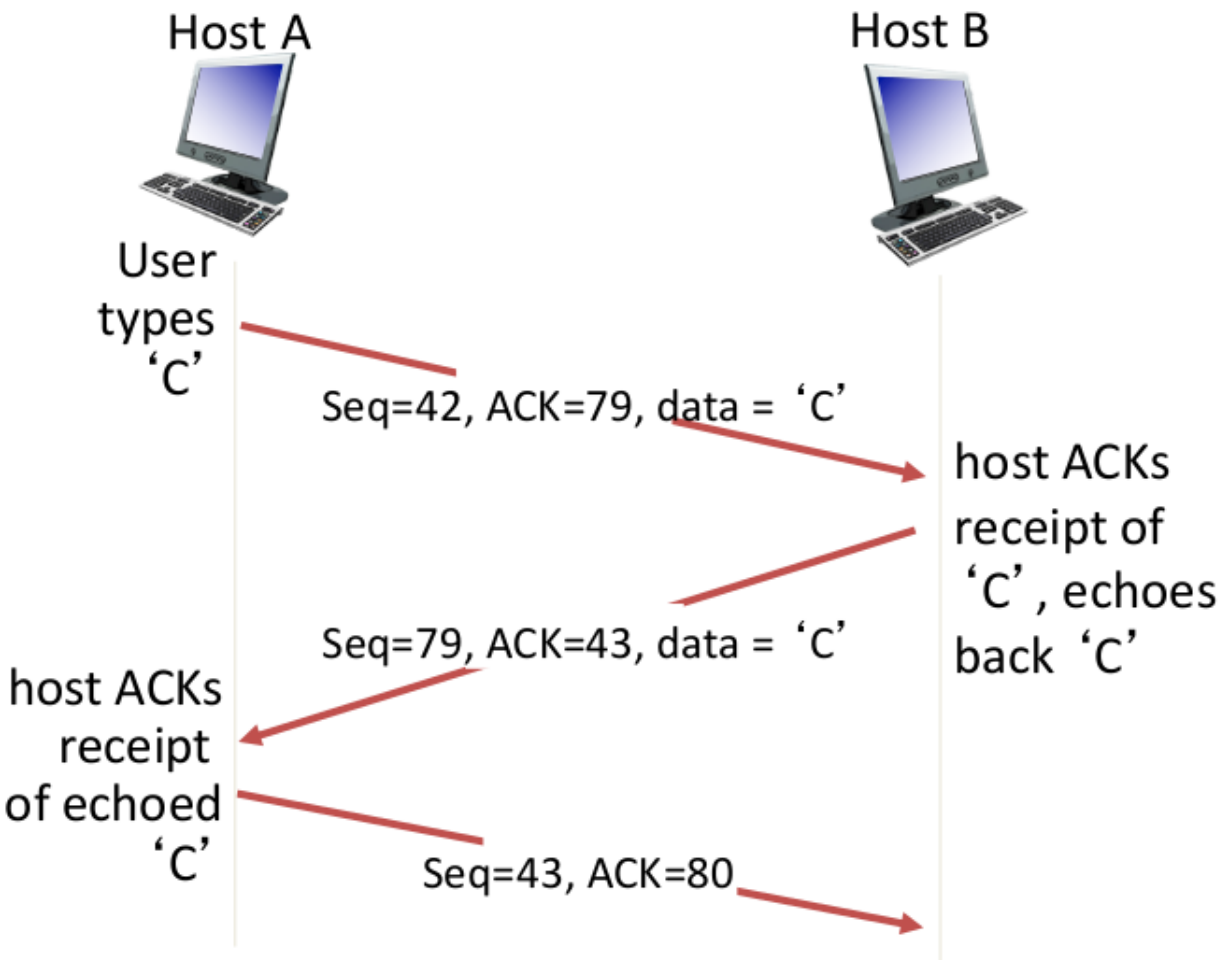
- Flags:
  * URG: Wichtige Daten sind schnell zu übertragen (Selten)
  * ACK: Bestätigung des Sendens
  * PSH: Push data – Daten sofort zur Anwendung pushen
  * RST, SYN, FIN: Verbindungsmanagement

    * Syn: Verbindungsaufbau – Synchronize
    * FIN: Verbindungsende
    * RST: Verbindungszzurücksetzung – Reset
- Urgent-Pointer: Mit URG-Flag.
- Options: Maximum Segment Size, z.B.
- Data: Anwendungsdaten

- SEQ & ACK-Nummern: Reihenfolge & Zuverlässigkeit.
    Seq: Nummer des ersten Bytes im Segment
    Ack: Nummer des nächsten erwarteten Bytestreams



Telnet-Example:



> Old ACK → New SEQ> OLD SEQnr + Länge → new ACK
> Frage: Wielange muss der Timeout von Host A sein?
→ Allgemeine uneinig.

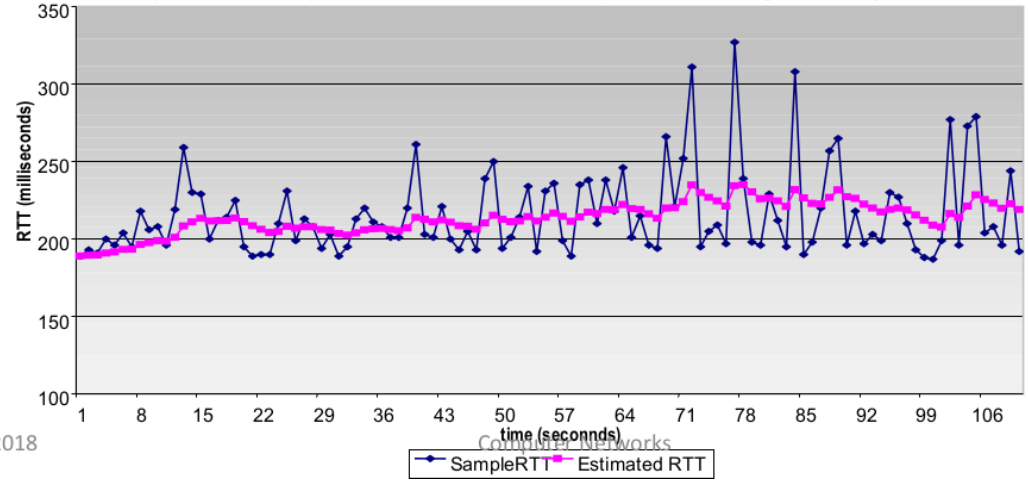
a.) Abschätzen

b.) EstimatedRTT

RTT must be estimated well – protocol based on time outs
Exponential weighted moving average (EWMA)

EstimatedRTT = (1- α)*EstimatedRTT + α*SampleRTT

Influence of past sample decreases exponentially fast (recomm.: α=0.125)



.. including some safety margin for higher accuracy:

SaMa = (1-ß) * SaMa + ß*|SampleRTT – Estimated RTT|

→ Estimate ß to be 0.25

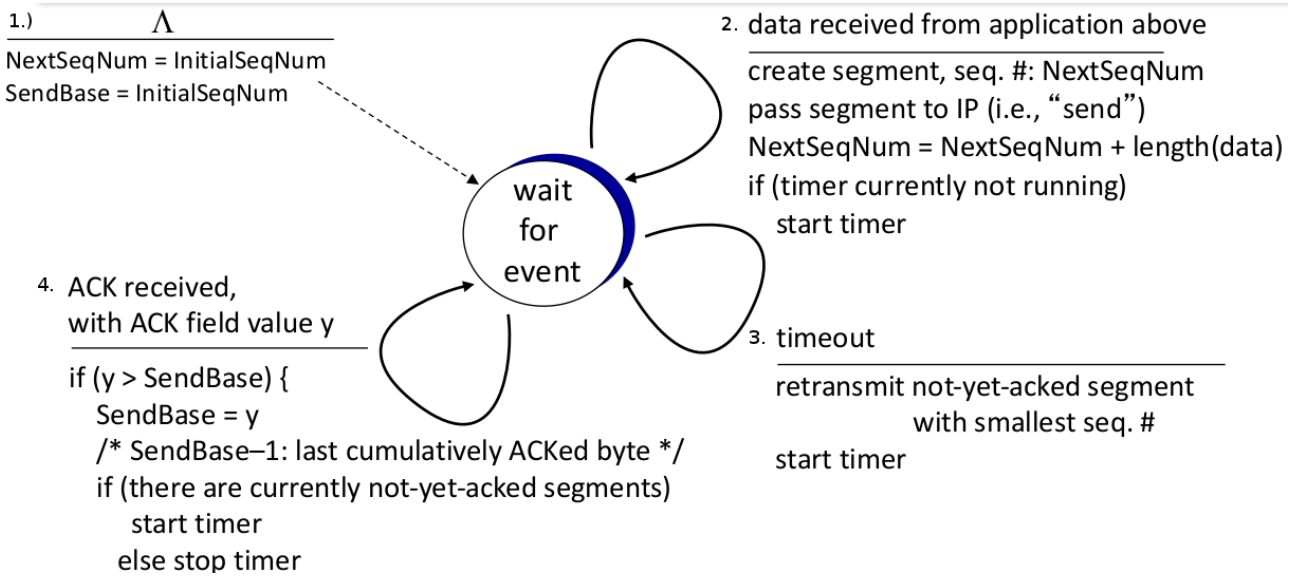
→ Timeout = EstimatedRTT + 4*SaMa

What the @Formels:

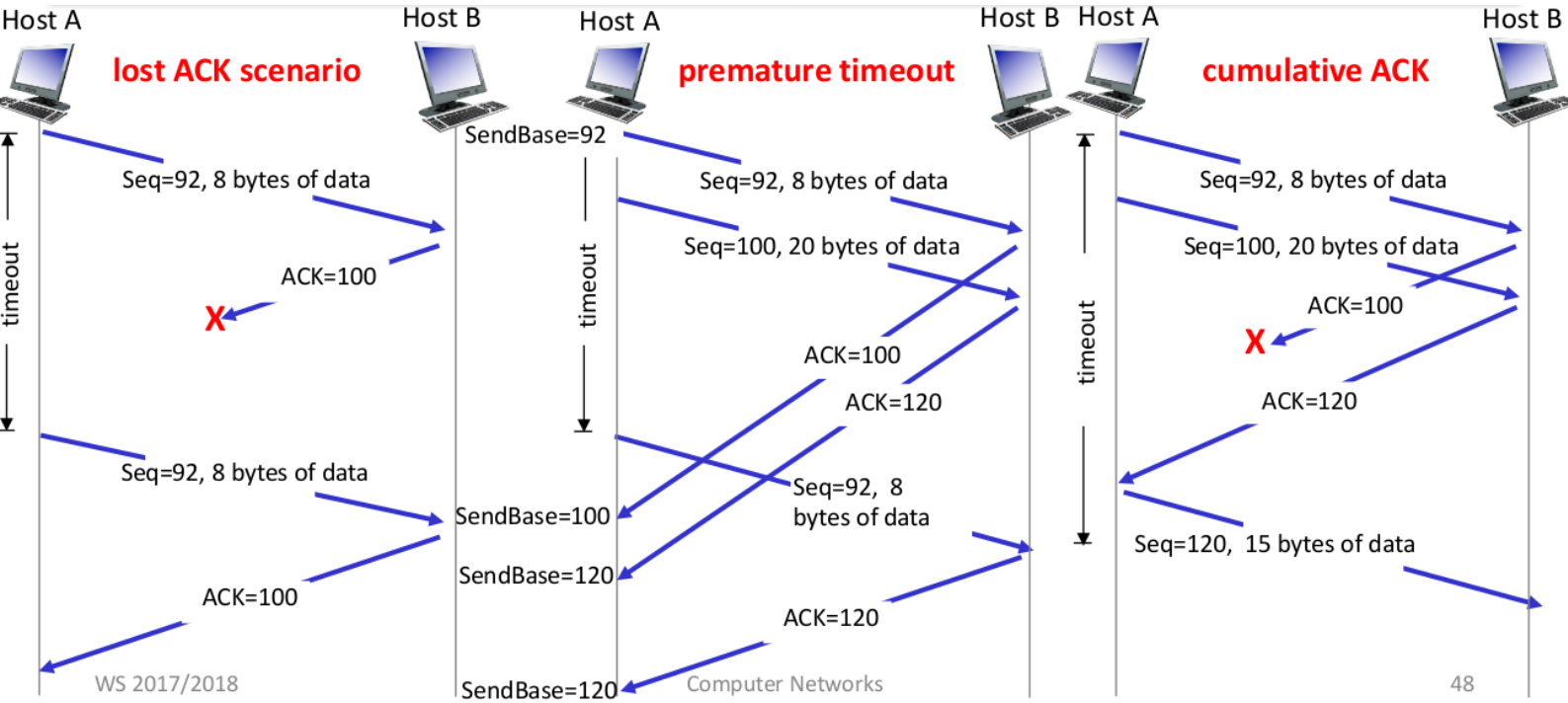
Aplha & Beta: Adaptiv änderbar; Angenommen aus best practice Praktiken

<u>TCP:Reliable Datatransfer:</u>

- Pipelined Segmets, Cumulative ACKs, Retransmission Timer
- Retransmissions triggered by Timeout or Duplicate ACKs



Example:

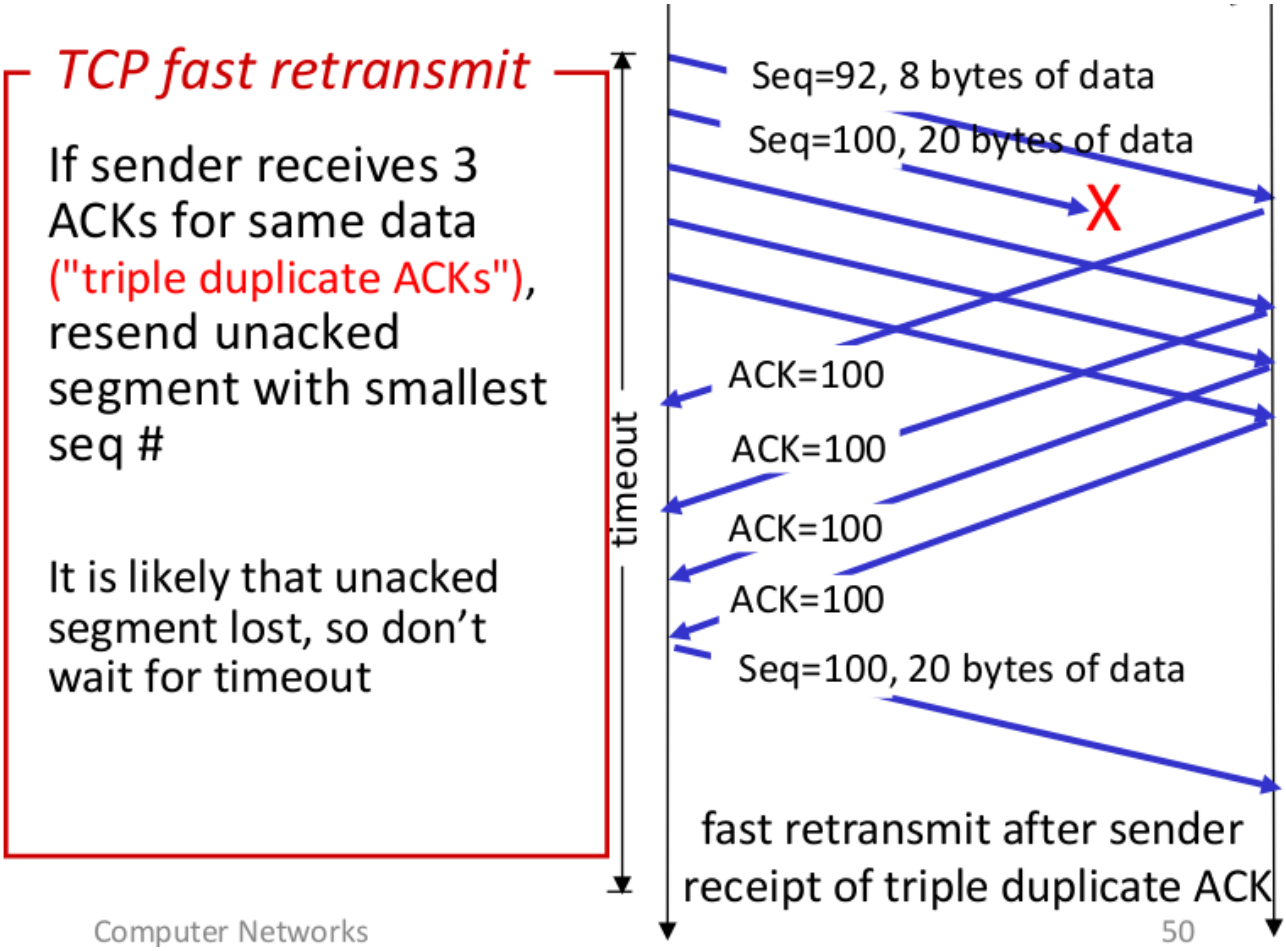


Ask 44

TCP ACK Generation [RFC 1122, RFC 2581]

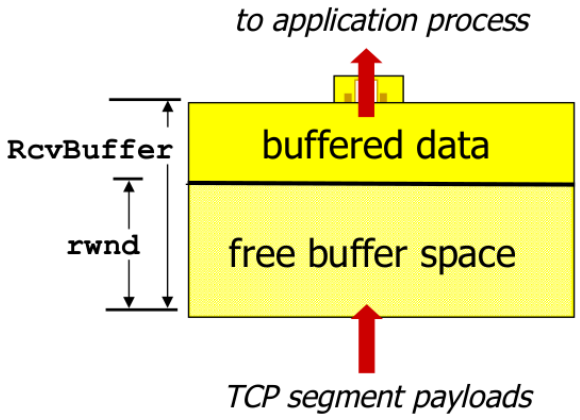
	event at receiver	*TCP receiver action*
Case 1	arrival of in-order segment with expected seq #. All data up to expected seq # already ACKed	delayed ACK. Wait up to 500ms for next segment. If no next segment, send ACK
Case2	arrival of in-order segment with expected seq #. One other segment has ACK pending	immediately send single cumulative ACK, ACKing both in-order segments
Case 3	arrival of out-of-order segment higher-than-expect seq. # . Gap detected	immediately send *duplicate ACK*, indicating seq. # of next expected byte
Case 4	arrival of segment that partially or completely fills gap	immediate send ACK, provided that segment starts at lower end of gap

<u>TCP Fast Transmit</u>:
Bei Timeouts: Alle Packete startend mit zuletzt bestätigter ACK werden neu geschickt und neuer Timeout wird gestartet.



Flow Control:
Daten die einfließen werden im Empfangsbuffer geladen.
Flow Control steuert den Sender, damit kein Datenüberfluss entsteht.

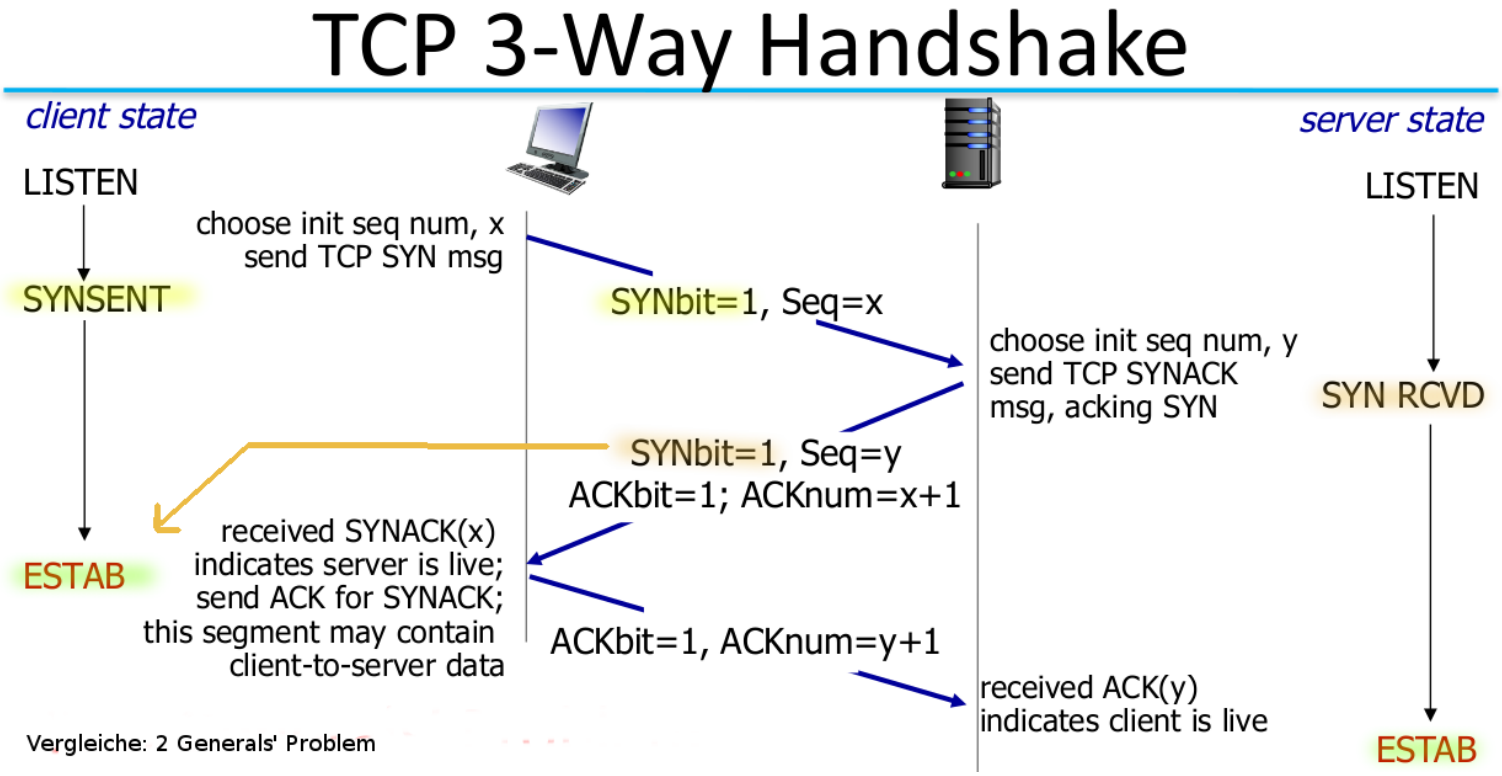
- Receiver "advertises" free buffer space by including **`rwnd`** value in TCP header of receiver-to-sender segments
  - **`RcvBuffer`** size set via socket options (typical default is 4096 bytes)
  - Many operating systems auto-adjust **`RcvBuffer`**
- Sender limits amount of unacked ("in-flight") data to receiver's **`rwnd`** value
- Guarantees receive buffer will not overflow



receiver-side buffering

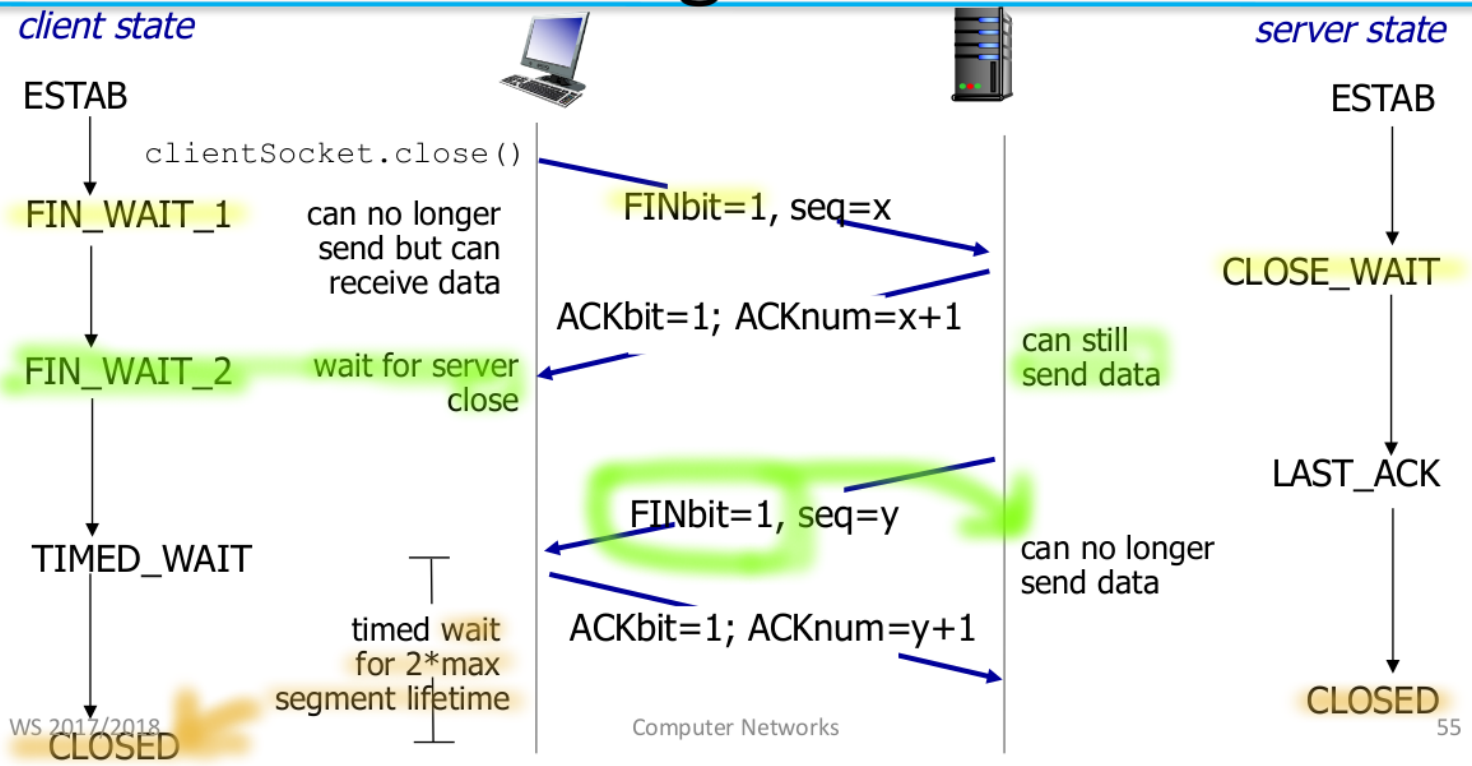
Flow-Control vs Congestion-Control – Kenne den Unterschied:Flow: Empfänger gibt an wieviele Daten verwendet werden kann (Pic right above)
Congestion:

<u>Connection Management</u>:
1. Handshake: Verbindungsaufbau via Verbindungsbestätigung beider Medien



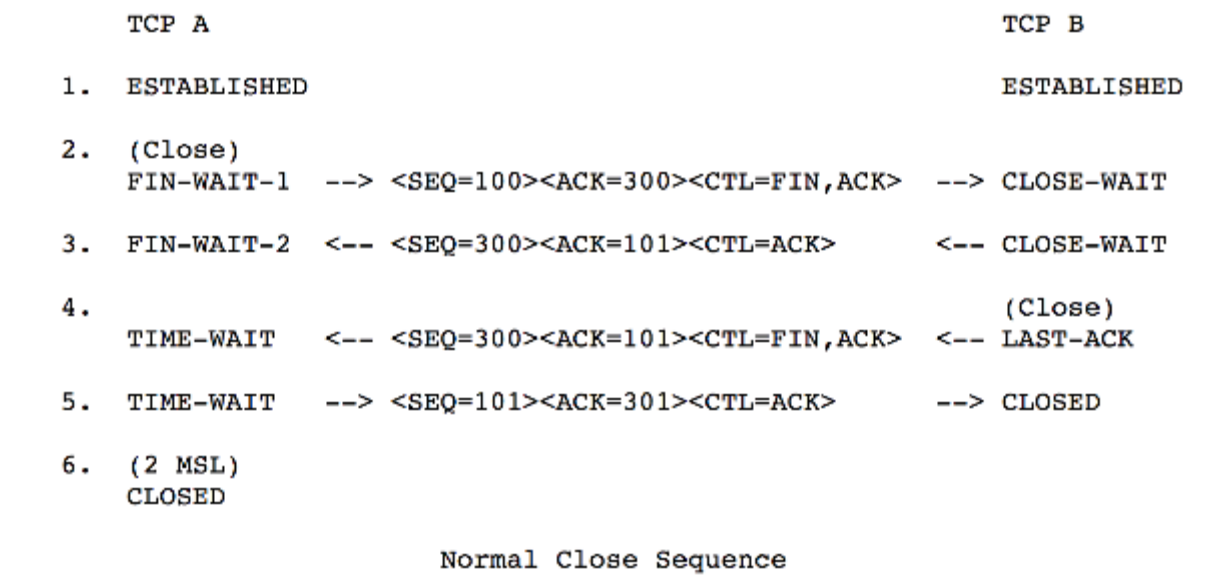
Optimistische Ansicht: Man geht nach diesem 3-way Handshake davon aus, dass die Verbindung tatsächlich besteht.

TCP Closing Connection

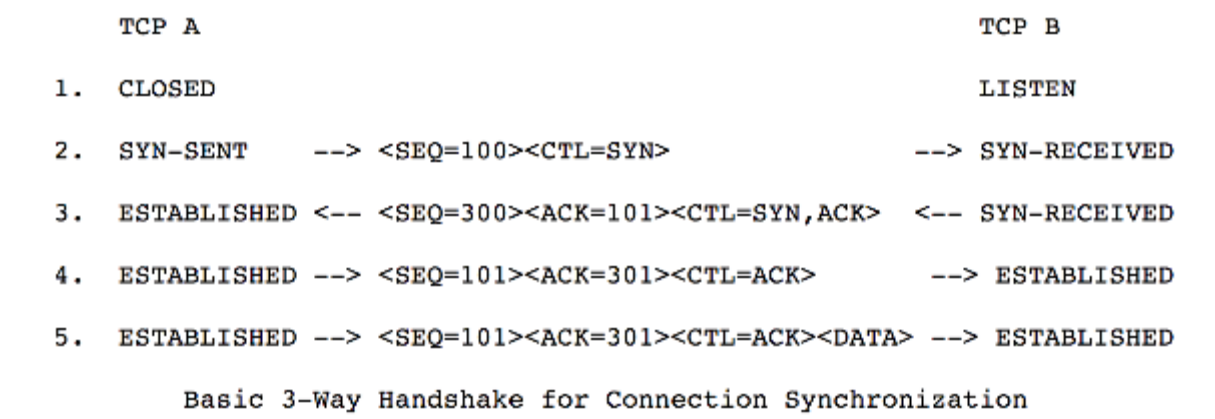


Abbau:

.. wie in RFCs beschrieben:



Normal Close Sequence

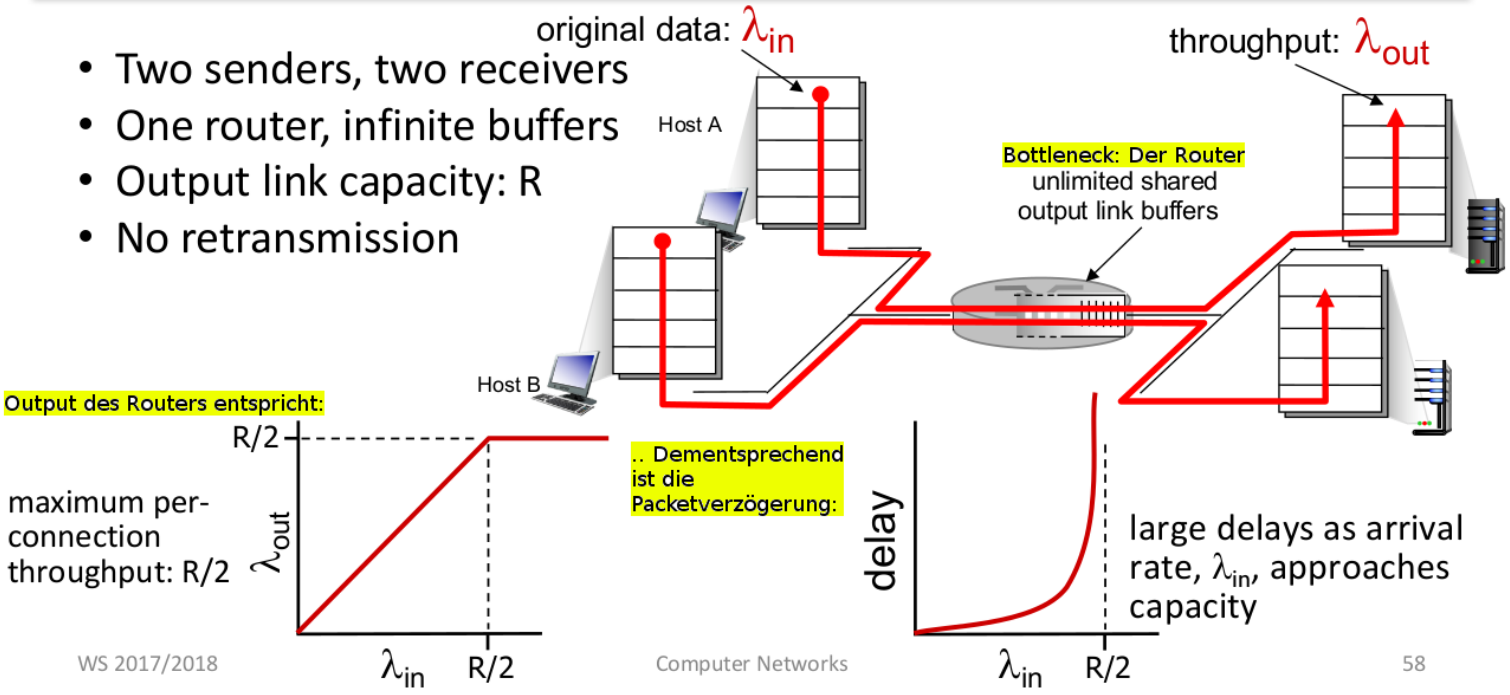


Basic 3-Way Handshake for Connection Synchronization

Die tatsächliche Übertragung wird nach wie vor auf Anwendungsschicht implementiert.

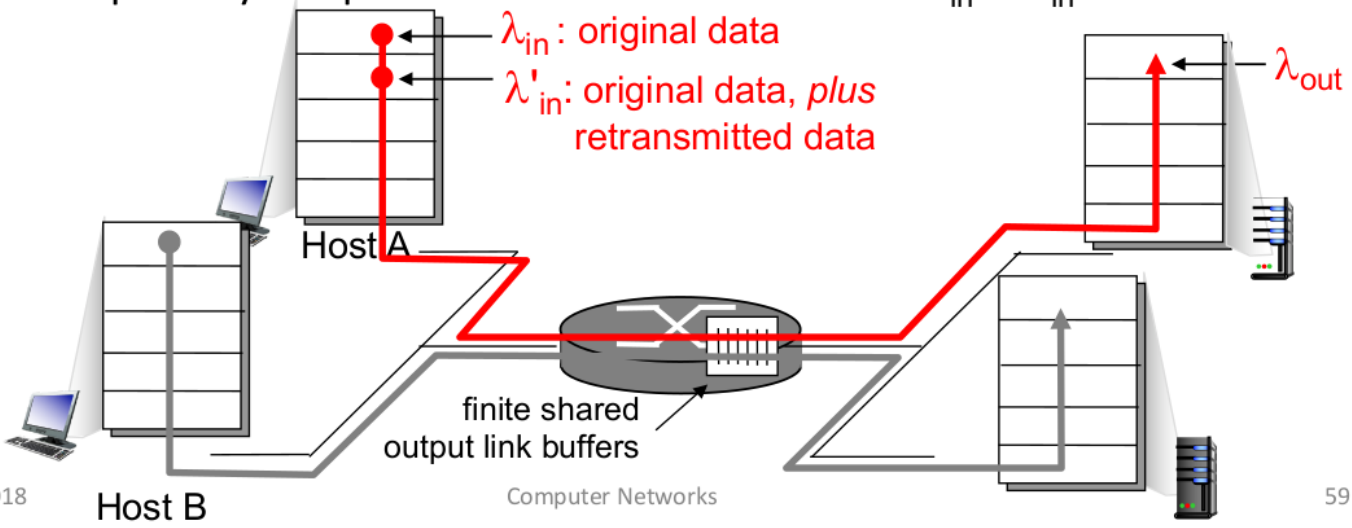
Congestion Control:
Viele Quellen schicken zuviele Daten
→ Es verwirft Daten auf Netzwerkschicht, IP-Layer.
Dementsprechend gibt es keine guten Mitteilugnsmöglichkeiten um die Senderate anzupassen.
- Rep.: Flow Control = Empfänger übermittelt Überlauf
Probleme:
1.) Packetverlust: Bufferoverflow in Router
2.) Verzögerungen: Queueing of Data ↔ End2End-Verbindungsdelay wird immer größer

- Two senders, two receivers
- One router, infinite buffers
- Output link capacity: R
- No retransmission

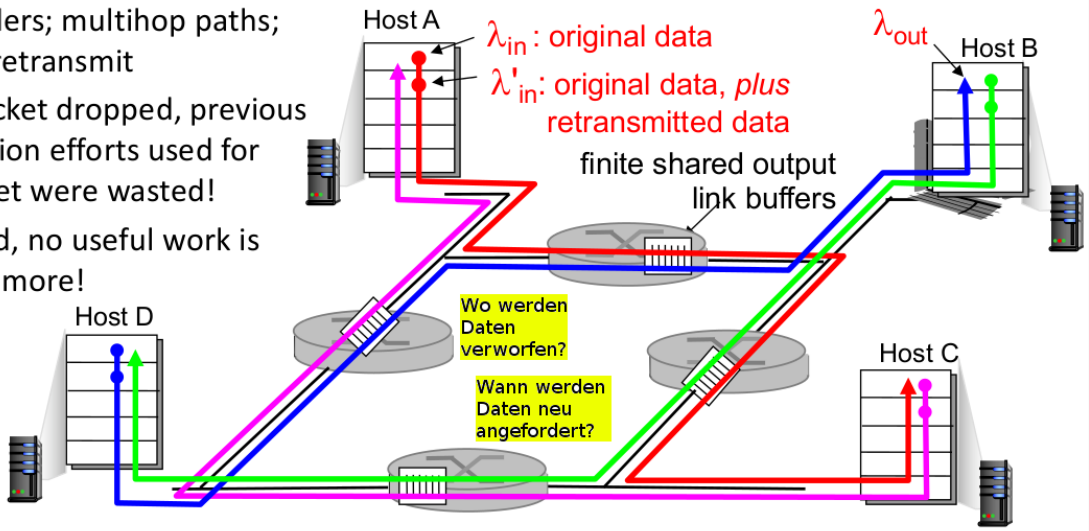


Annahme des unendlichen Buffers.
- Wäre der Buffer jedoch voll, so muss es ein Retransmission geben → NOCH mehr Packete
→ e.g. Tatsächliche Übertragung kann auf 1/3 bis ¼ der üblichen Latenz gehen:
→ e.g. Es gibt duplikate innerhalb des Transfers
somit gibt es Throughput-Verlust.

- One router, *finite* buffers
- Sender retransmission of timed-out packet
  - Application-layer input = application-layer output: $\lambda_{in} = \lambda_{out}$
  - Transport-layer input includes *retransmissions* : $\lambda'_{in} \geq \lambda_{in}$



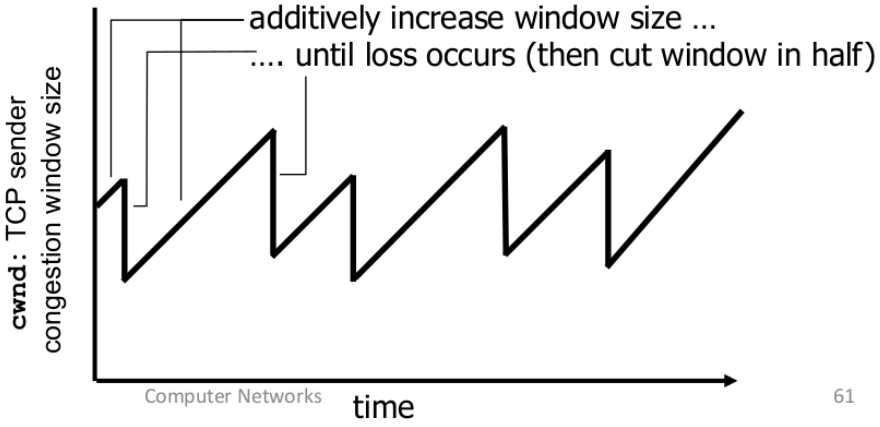
- Four senders; multihop paths; timeout/retransmit
- When packet dropped, previous transmission efforts used for that packet were wasted!
- At the end, no useful work is done any more!



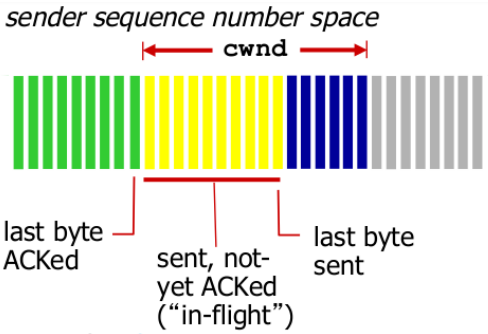
AIMD - Additive Increase Multiplicate Decrease:
Sender erhöht Schrittweise die Übertragungsrate, bis es ein Packetverlust eintritt. Dann halbiere die Übetragungsrate. Rinse & Repeat.

- Additive increase: increase **cwnd** by 1 MSS every RTT until loss detected
- Multiplicative decrease: cut **cwnd** in half after loss

AIMD saw tooth behavior: probing for bandwidth



Details++:



- Sender limits transmission:

$$\texttt{LastByteSent} - \texttt{LastByteAcked} \leq \texttt{cwnd}$$

- **cwnd** is dynamic, function of perceived network congestion

TCP sending rate:

- *Roughly:* send **cwnd** bytes, wait RTT for ACKS, then send more bytes

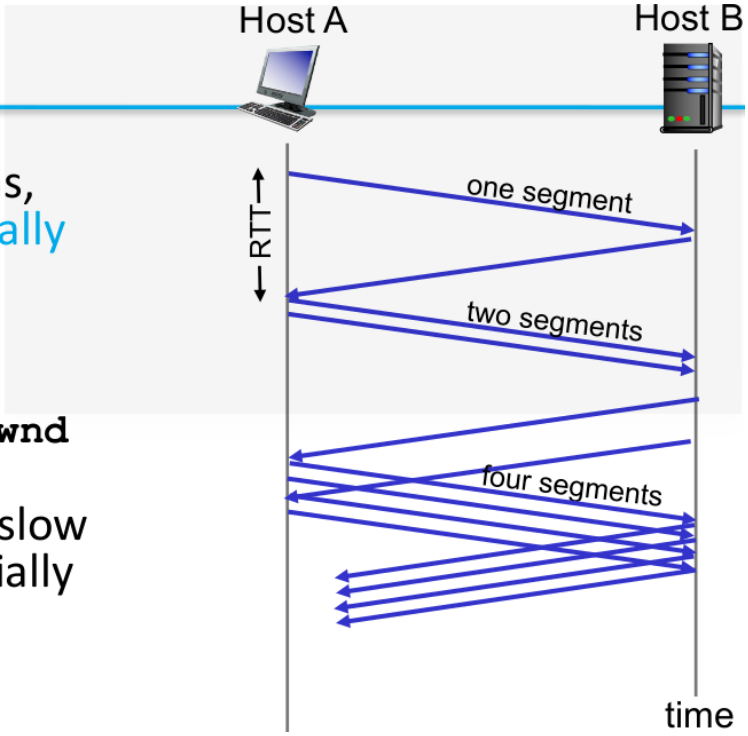
$$\text{rate} \approx \frac{\text{cwnd}}{\text{RTT}} \text{ bytes/sec}$$

Je nach darunterliegendem IpvX: Wir schicken soviele Bytes wie im congestion Window angegeben. Wenn Voll → Warte bis ACK kommt. Dann schicke weitere Daten.

Initial ist cwnd: 0. Er schickt dementsprechend 0^1 = 1 raus. Schrittweise erhöhung aka "Slow Start":

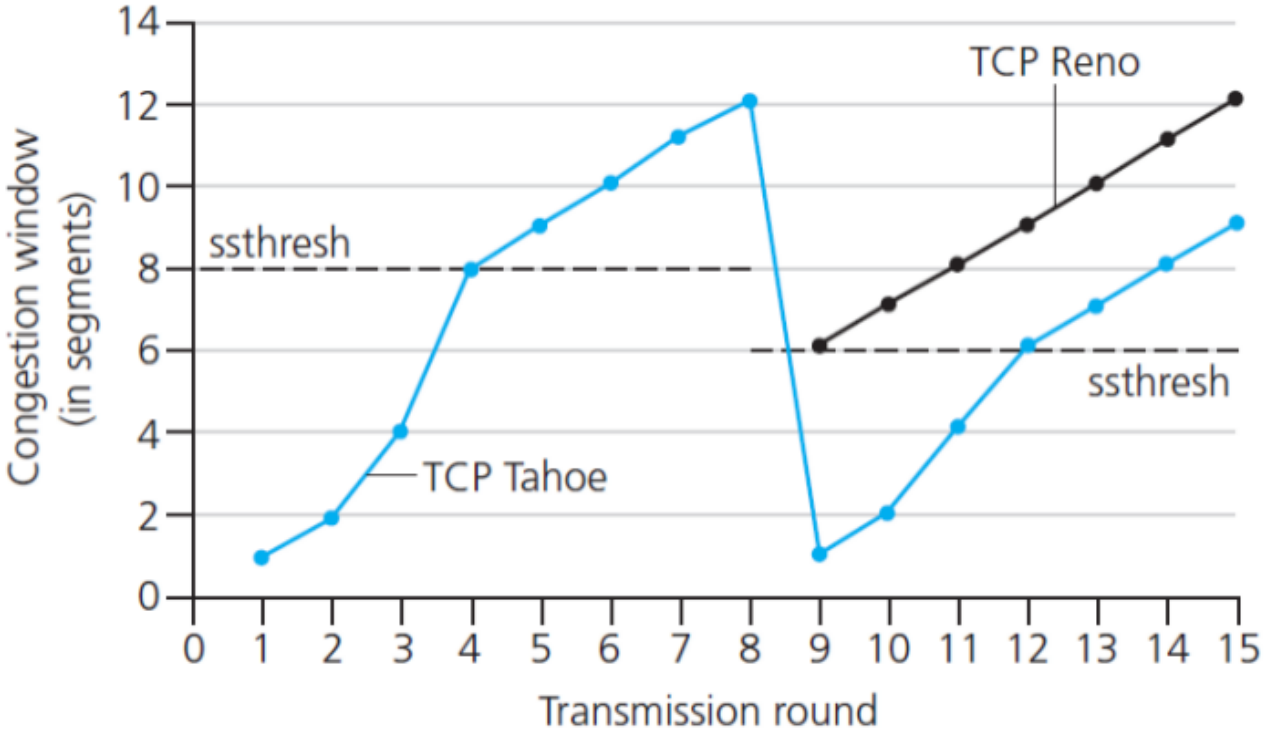
TCP Slow Start

- When connection begins, increase rate exponentially until first loss event:
  - Initially **cwnd** = 1 MSS
  - Double **cwnd** every RTT
  - Done by incrementing **cwnd** for every ACK received
- Summary: initial rate is slow but ramps up exponentially fast



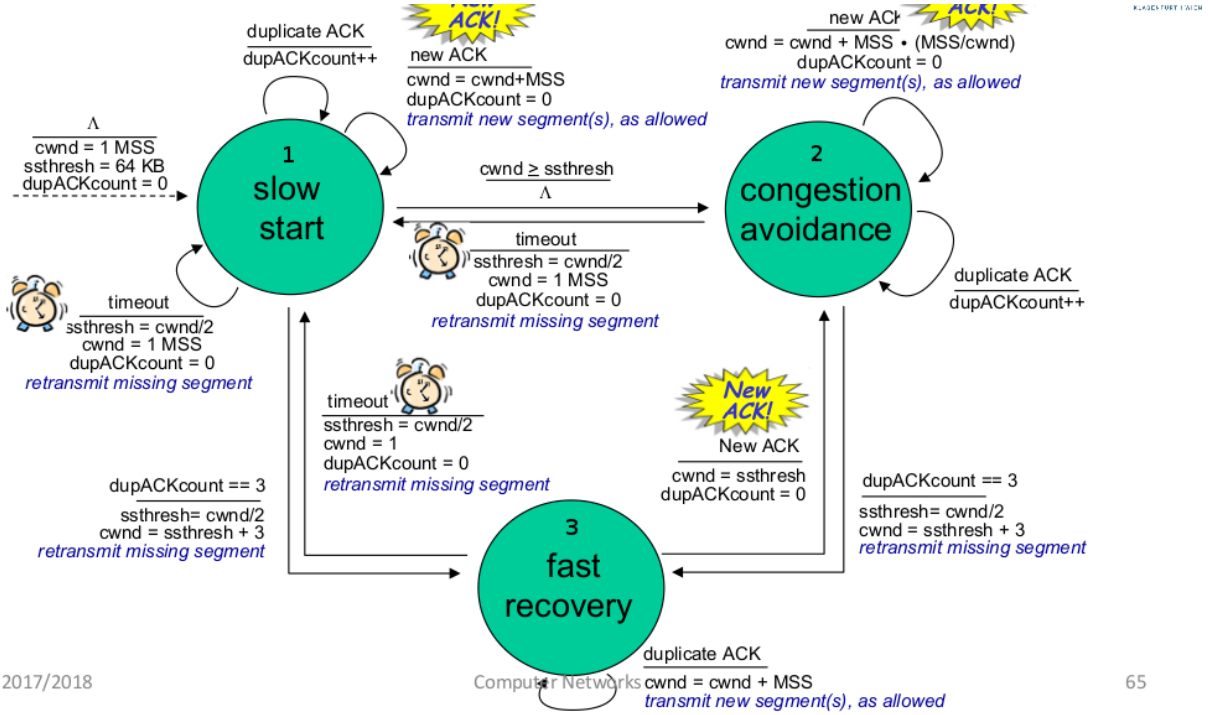
Dies hängt stark von der Implementierung ab.
Wird ein Loss detected, so:
- Schreite linear weiter (TCP RENO)

- Setze cwnd auf 1 zurück bis zu einem gewissen treshhold und steige linear weiter. (TCP Tahoe)

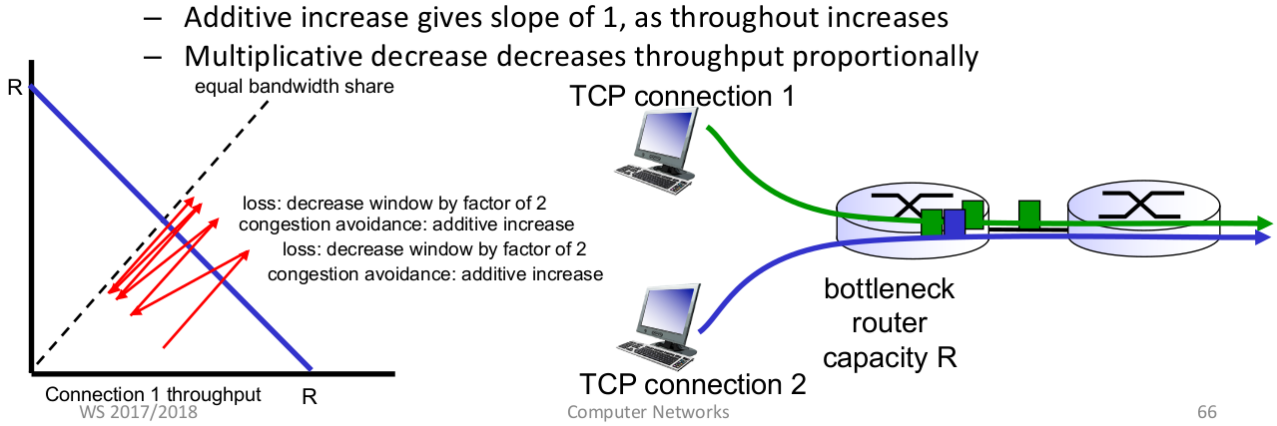


BSP.: StreamingJedes Video wird in Teile zerlegt, jedes Segment hat einige MB welches gesendet wird, via eigenem HTTP-Request.
Bei nicht-persistenter Verbindung: Verbindung → GET → Verbindungsaufbau → Verbindung → …
E.g. *Wir kommen nie auf die tatsächliche Übertragungsrate*.
Implementierungstechnisch könnte man z.B. nicht von 0 aus starten, sondern Risikoreich mit 10 o.ä. .

Prinzipiell ist TCP:



<u>TCP Fairness</u>:
Jeder sollte [R/K = Bandbreite/TCP-Sessions] an Datenübertragung zugesprochen bekommen.
→ Dabei hilft das AIMD, im Idealfall. Es gibt allerdings nicht nur TCP als Verbindungsprotokoll.



UDP:
- Multimedia Anwendungen nutzen oftmals UDP
→ Not throttled by congestion control.
Behandlung: Übertragung via konstanter, verlust-toleranter Rate.

- Parallelle Verbindugen zwischen 2 Geräten
→ Eine App erhöht den Durchsatz des eigenen Gerätes, VGL. Formel R/K

Fairness and UDP

- Multimedia apps often do not use TCP
  - Do not want rate throttled by congestion control
- Instead use UDP:
  - Send audio/video at constant rate, tolerate packet loss

Fairness, parallel TCP connections

- Application can open multiple parallel connections between two hosts
- Web browsers do this
- e.g., link of rate R with 9 existing connections:
  - new app asks for 1 TCP, gets rate R/10
  - new app asks for 11 TCPs, gets R/2

ECN - Explicit Congestion Notification:
Bei Problem, so kann man im IP-Header 2 Bits setzen, das ECN-Bit.
Das Ziel merkt, es gibt ein Problem und sendet ECE=1 zurück.
– Relativ neu.

Network-assisted congestion control:

- Two bits in IP header (ToS field) marked by network router to indicate congestion
- Congestion indication carried to receiving host
- Receiver (seeing congestion indication in IP datagram) sets ECE bit on receiver-to-sender ACK segment to notify sender of congestion