TCP: Congestion Avoidance and Flow Control

Hauke Kaufhold

2.12.14

TCP-Flußkontrolle (1)

- Menge der Daten muß an die Aufnahmefähigkeit des Empfängers angepaßt werden
- Abstimmung wird *Flußkontrolle (flow control)* genannt
- TCP verwendet dazu das Prinzip des Sliding Window
- Ziel: maximale Performance und Übertragungssicherheit in heutigen Netzen (RFCs 1122, 1323, 2001, 2018)

TCP-Flußkontrolle (2): benutzte Variablen

- *Sequence Number (Sequenznummer)*:
- → Fortlaufende Nummerierung der gesendeten Segmente

- *Acknowledgement Number (Quittungs- / Bestätigungsnummer)*:
- → Bestätigte erhaltene Segmente

- *Window-Größe*:
- → Maximale Anzahl an Datensegmenten, die vom Empfänger aufgenommen werden können (Bsp: 4)

TCP-Flußkontrolle (3): fehlerfreies Beispiel

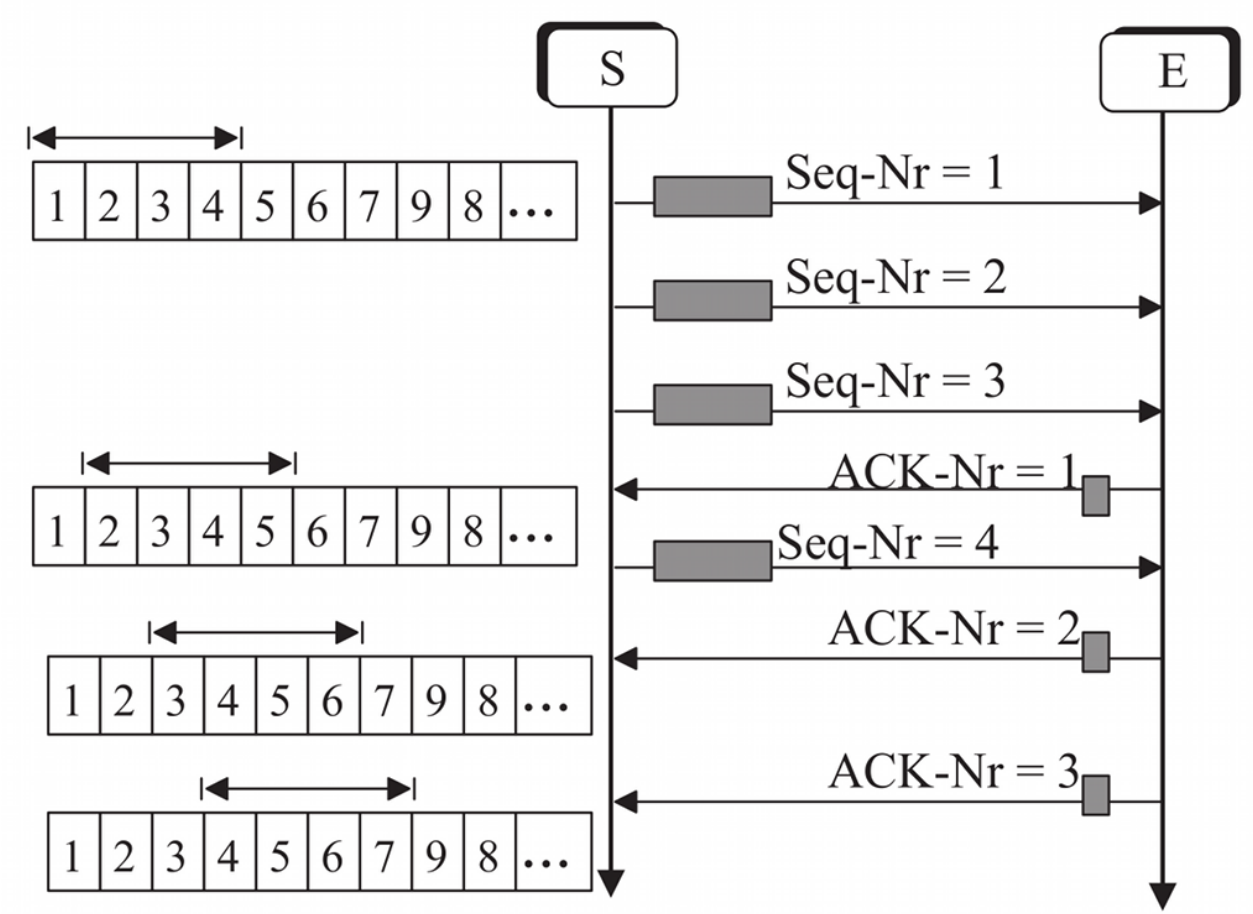

Abbildung: Quelle: [2], Termin 10

TCP-Flußkontrolle (4)

- Ablauf nicht immer fehlerfrei
- *Sendeblockaden* sind möglich, wenn (z.B.)
  - Window-Größe zu klein ist
  - Verzögerungszeit im Netz zu groß ist
- Sender muss dann auf Quittung warten und blockiert

TCP-Flußkontrolle (5): blockiertes Beispiel

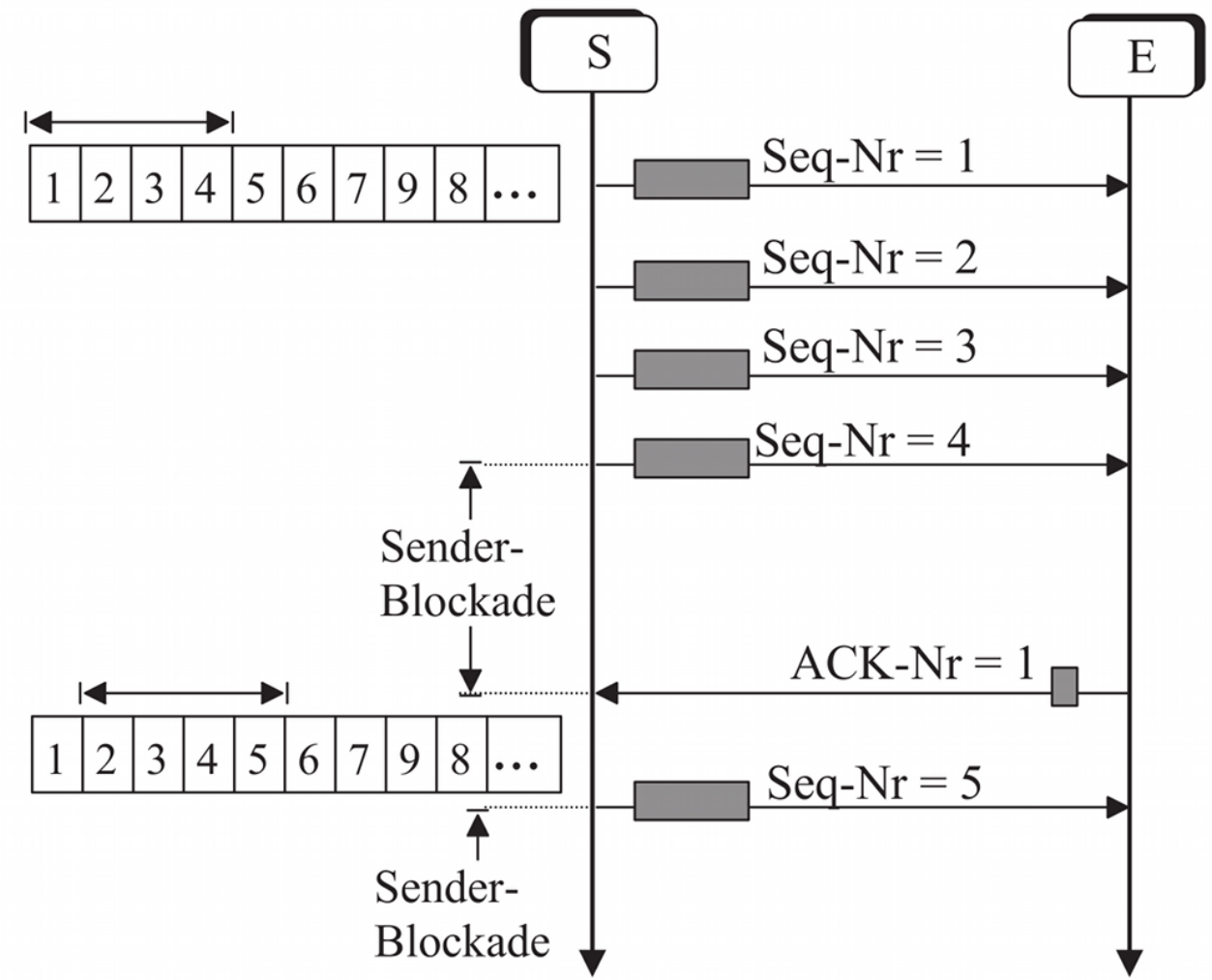

Abbildung: Quelle: [2], Termin 10

TCP: Sliding Window (1)

- TCP benutzt eine modifizierte Version des Sliding Window-Prinzips
- Window-Größe wird in Bytes angegeben (nicht in Segmenten)
- Erlaubt unterschiedlich große Segmente
- Empfangspuffer hat nur noch eine Puffergröße (keine Segmentanzahl)

TCP: Sliding Window (2)

Drei mögliche Szenarien:

1. fehlerfreie Datenübermittlung
2. Sendeblockade bei der Datenübermittlung
3. fehlerbehaftete Datenübermittlung

TCP: Sliding Window (3) - Wartezeit

- Maximale Wartezeit ist ein wichtiger Parameter im TCP-Betrieb
- Hängt u.a. von der Verzögerung ab (Messung der RTT)
- Wichtig für Retransmission
- Variiert im Laufe der Zeit
- $\rightarrow$ dynamische Berechnung von Nöten
- Verschiedene Verfahren werden zur Optimierung eingesetzt (Nagle, Karn/Partridge, Jacobsen/Karel)

Nagle-Algorithmus

- Ziel: Verhinderung vom Senden kleinster Segmente
- Nagle-Algorithmus berücksichtigt 4 Faktoren:
  - TCP-Haltezeit für das Zusammenführen von Applikationsdaten in Segmente
  - Verfügbarer TCP-Pufferbereich für Applikationsdaten
  - RTT
  - Applikationstypen (besonders kritisch sind interaktive Protokolle (Tastatureingaben, etc))

Silly Window Syndrome

- Komplementärer Effekt zum Nagle-Algorithmus
- Kann bei großen Datenmengen vorkommen
- Empfänger erlaubt senden (Acks) obwohl freier Puffer nur sehr klein ist (Window Size)
- Kann vermieden werden, wenn der Empfänger mit dem ACK lange genug wartet, bis genug Puffer frei ist

Karn/Partridge-Implementierung

- Dient der Abschätzung der RTT
- Wartet auf ACK vom Emfpänger
- Wiederholte TCP-Pakete werden nicht betrachtet
- Timeout-Zeit wird nach Auslauf angehoben (verdoppelt)
- $\mathit{Timeout}' := 2 \times \mathit{Timeout}$

Congestion Control (1)

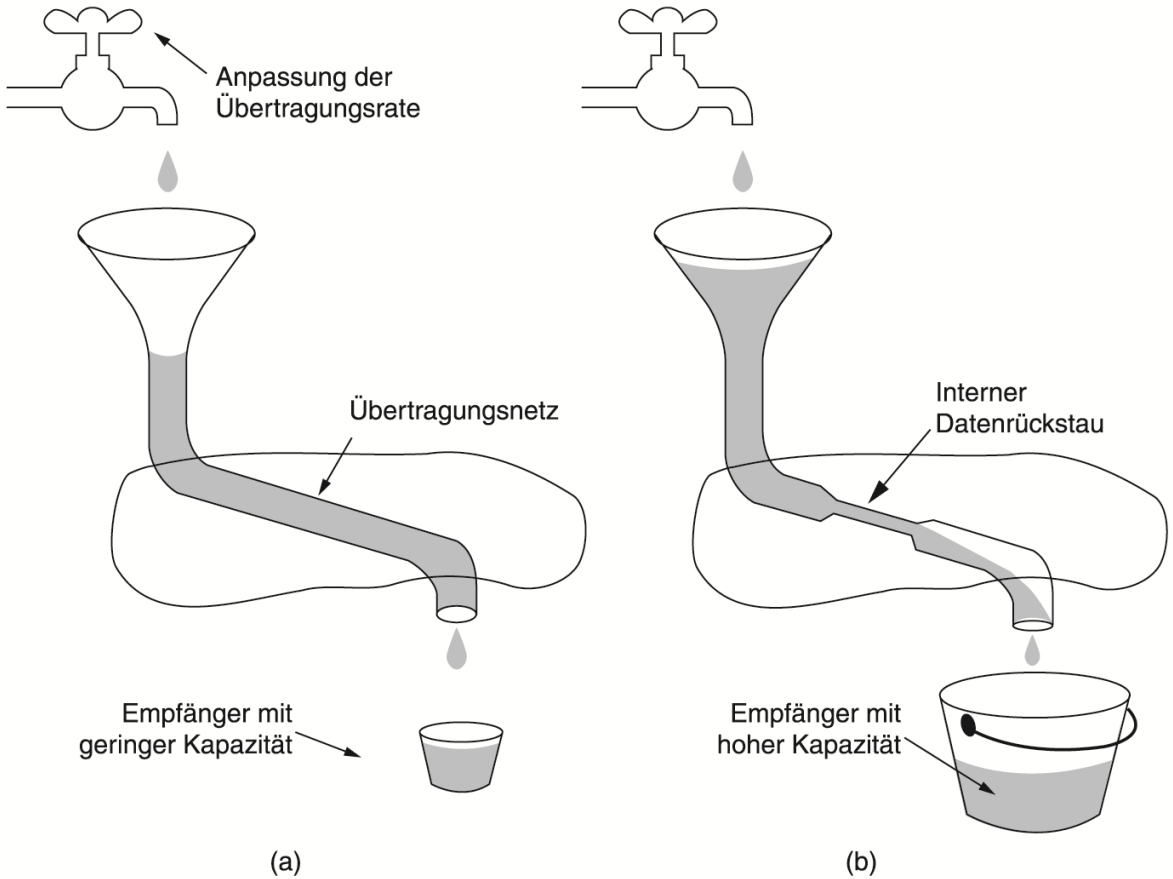

Abbildung 6.36: (a) Ein schnelles Netz, das an einen Empfänger mit geringer Kapazität weiterleitet (b) Ein langsames Netz, das an einen Empfänger mit hoher Kapazität weiterleitet

Abbildung: Quelle: [1], Seite 597

Congestion Control (2)

- *Congestion Avoidance*: Datenpakete gehen kaum noch verloren (Annahme)
- Hauptgrund für Timeouts und doppelt empfangener ACKs (dACKs): Überlastung im Netzwerk
- RFC 2001
- Wichtige Variablen:
  - cwnd (*Congestion Window*)
  - ssthresh (*Slow Start Threshold*)
  - advWindow (*Advertised (Receiver) Window*)

Congestion Control (3): Slow Start + Congestion Avoidance

- Initialisiere $cwnd = 1$, $sstresh =$ max Windowsize (64 Kbyte)
- Maximale zu sendende Datenmenge: $min(cwnd, advWin)$
- Bei Empfang von dACKs: exponentielle Vergrößerung von *cwnd*
- Bei Timeout: $sstresh = 0{,}5 \times$ cwnd(alt), $cwnd = 1$ (Slow Start)
- Falls $cwnd \geq sstresh \rightarrow$ lineare Vergrößerung von *cwnd*
- Falls $cwnd = advWin \rightarrow$ konstant halten

Congestion Control (4) - (Beispiel)

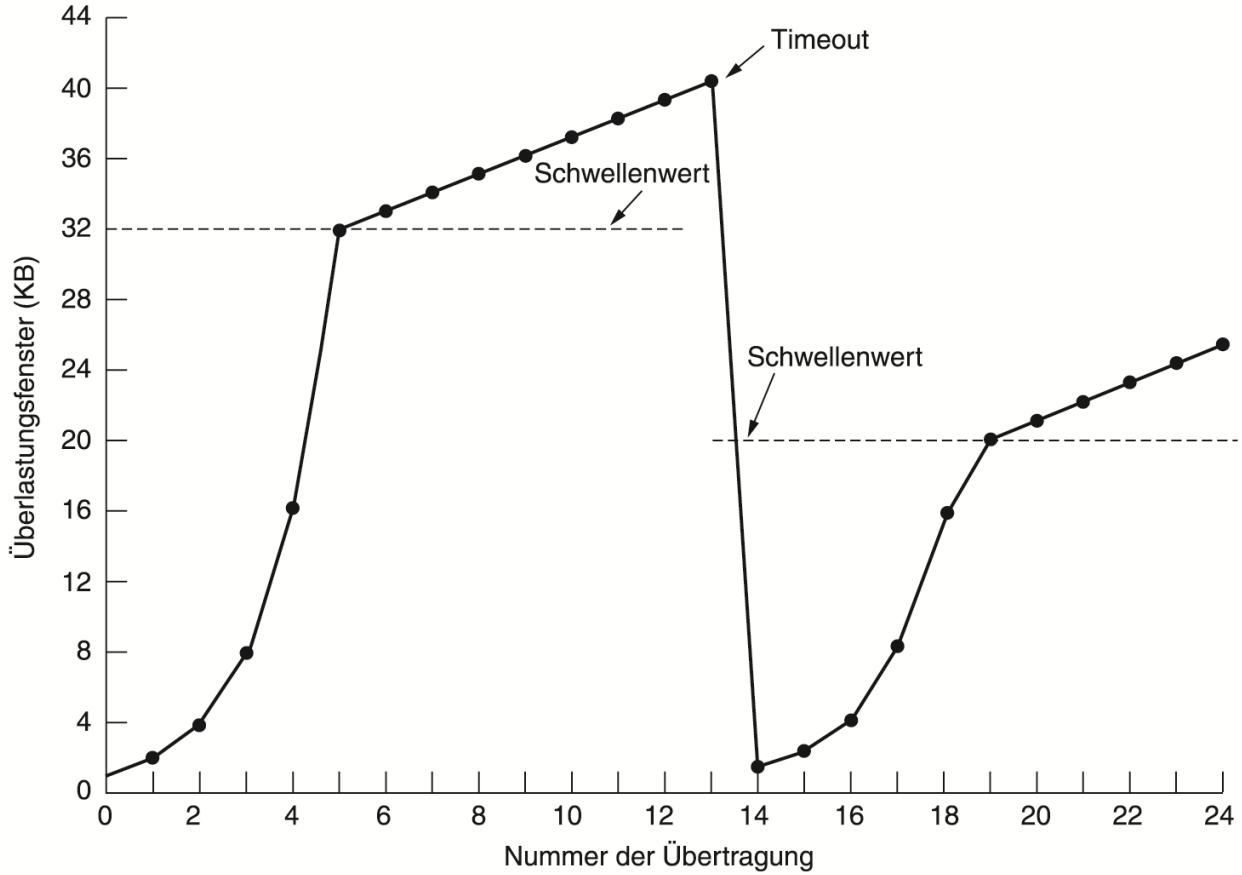

Abbildung 6.37: Beispiel des Überlastungsalgorithmus im Internet

Abbildung: Quelle: [1], Seite 599

Congestion Control (5)

- *Fast Retransmit/Fast Recovery*: Annahme, dass mehre dACKs nur den Verlust eines Segmentes bedeuten
- Falls ein Schwellwert (z.B. 3dACKs) überschritten wird
  $\rightarrow$ erneutes senden, ohne auf Timeout zu warten
- Danach Fast Recovery: Annahme, dass noch ein anhaltender Datenstrom vorliegt

Congestion Control (6)

- *Selective Acknowledgements*: Empfänger teilt Beginn und Ende der zuletzt zusammenhängend empfangen Datenblöcke mit
- Verhindert, dass mehrere TCP-Segmente wiederholt versendet werden würden
- Fast Retransmit würde hingegen mehrere TCP-Segmente wiederholen
- SACK-Option erweitert das Optionsfeld

Fragen?

Fragen?

Danke

Danke für eure Aufmerksamkeit!

Quellen

 [TCN] Tanenbaum, Andrew S; Computernetzwerke; 4. Auflage; Pearson Studium 2003

 [DKI] Sieker, Holtkamp et al; Digitale Kommunikation und Internetdienste; Universität Bielefeld, WS 2013