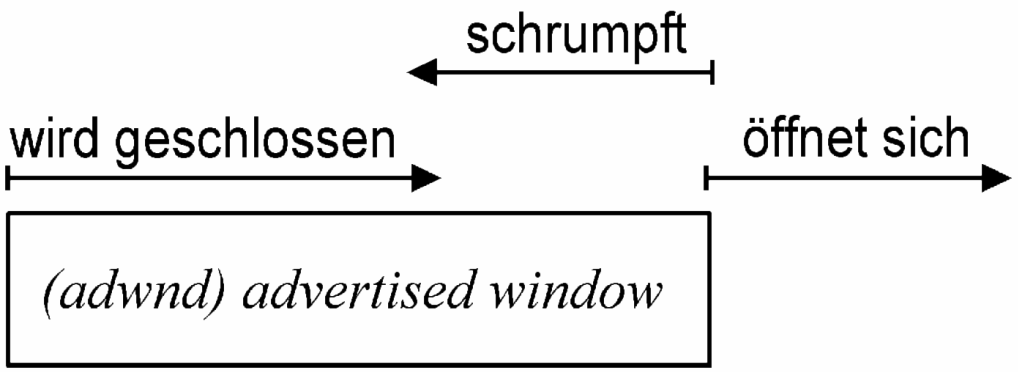
# TCP Teil 2

- sliding window protocol
- Begriffe: MSS, RTT und RTO
- bulk-data flow
- Stau-Vermeidung
- Langsamer Start
- Zusammenspiel: S.V. und L.S.
- TCP features und options

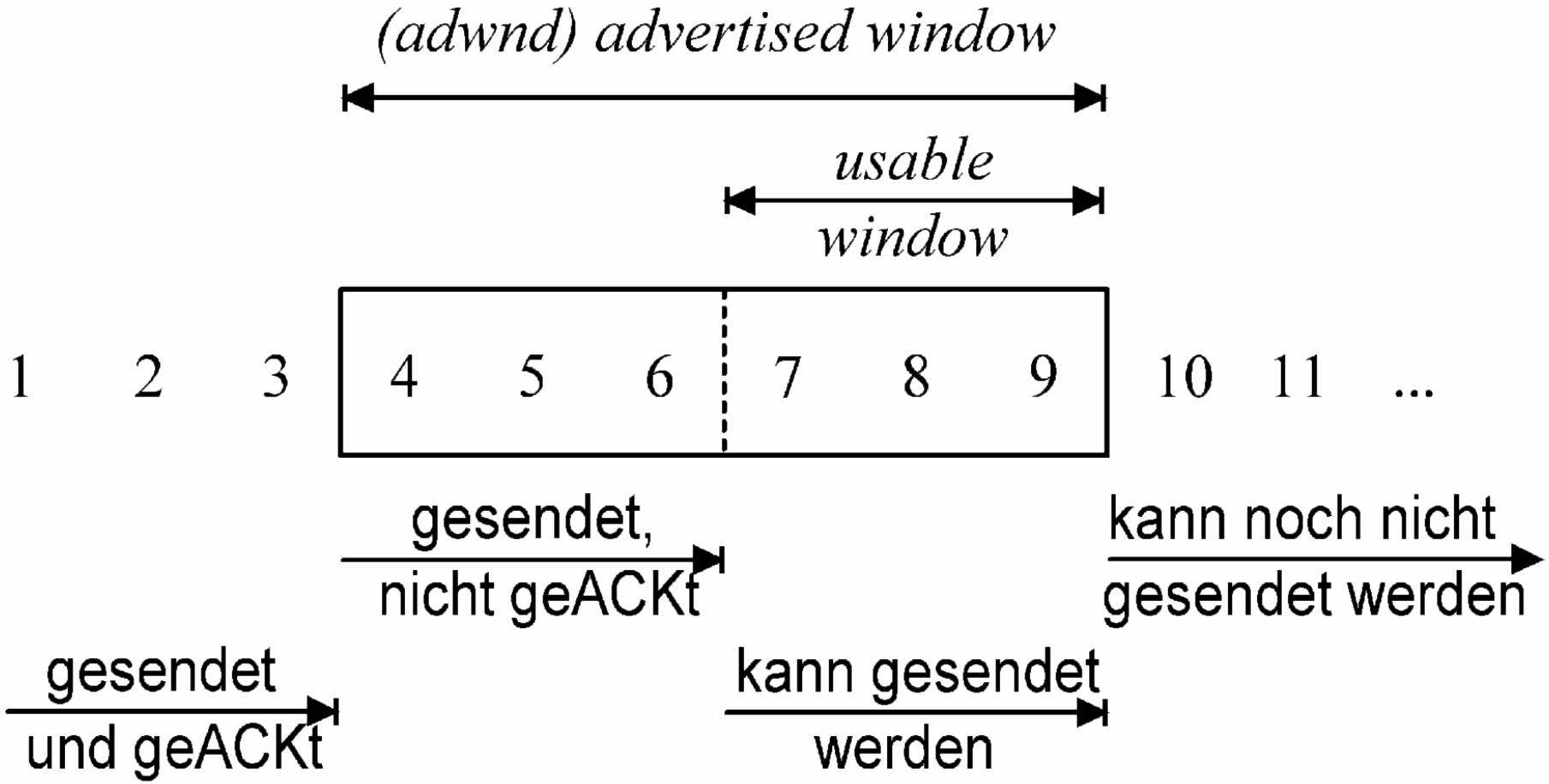
# sliding window protocol

**Grundidee:** Empfänger stellt ein „Fenster“ zu Verfügung, in dem Daten übertragen werden.
Die Größe des Fensters wird vom Empfänger gewählt, üblicherweise so groß wie der Empfangspuffer.




- **Öffnet sich**, wenn der Empfangspuffer geleert wird.
- **Wird geschlossen**, wenn Daten übertragen und geACKt werden.

# sliding window protocol



# Begriffe: MSS, RTT und RTO

• *MSS (maximum segment size)*
*M*aximale Segmentgröße, die TCP empfangen kann.
Hardware abhängig, berechnet aus der *MTU*

• *RTT (round-trip time)*
Zeit, die es dauert bis ein Datensegment vom Sender zum Empfänger und das zugehörige ACK zurückübertragen wird.
Die *RTT* wird für jede Verbindung dauernd neu gemessen.

• *RTO (retransmission timeout)*
Zeitintervall, das TCP (beim ersten Versuch) auf ein ACK wartet bevor das Segment erneut versendet wird.
Der *RTO* wird aus den *RTT*-Werten berechnet.

# bulk-data flow

- Versendung großer Datenmengen (z.B. bei ftp)

- Es gibt keine „normale“ Art und Weise der Datenübertragung
- Datenübertragung hängt ab von:
  - TCP-Implementierungen auf beiden Seiten
  - Schreiben und Lesen der Applikationen
    → Prozessverwaltung des OS
  - Den unteren Ebenen im Schichten-Modell.
  - Der Netzwerkdynamik auf der Verbindung

- Beispiel: Übertragung von 8192 bytes von „svr4“ nach „bsdi“

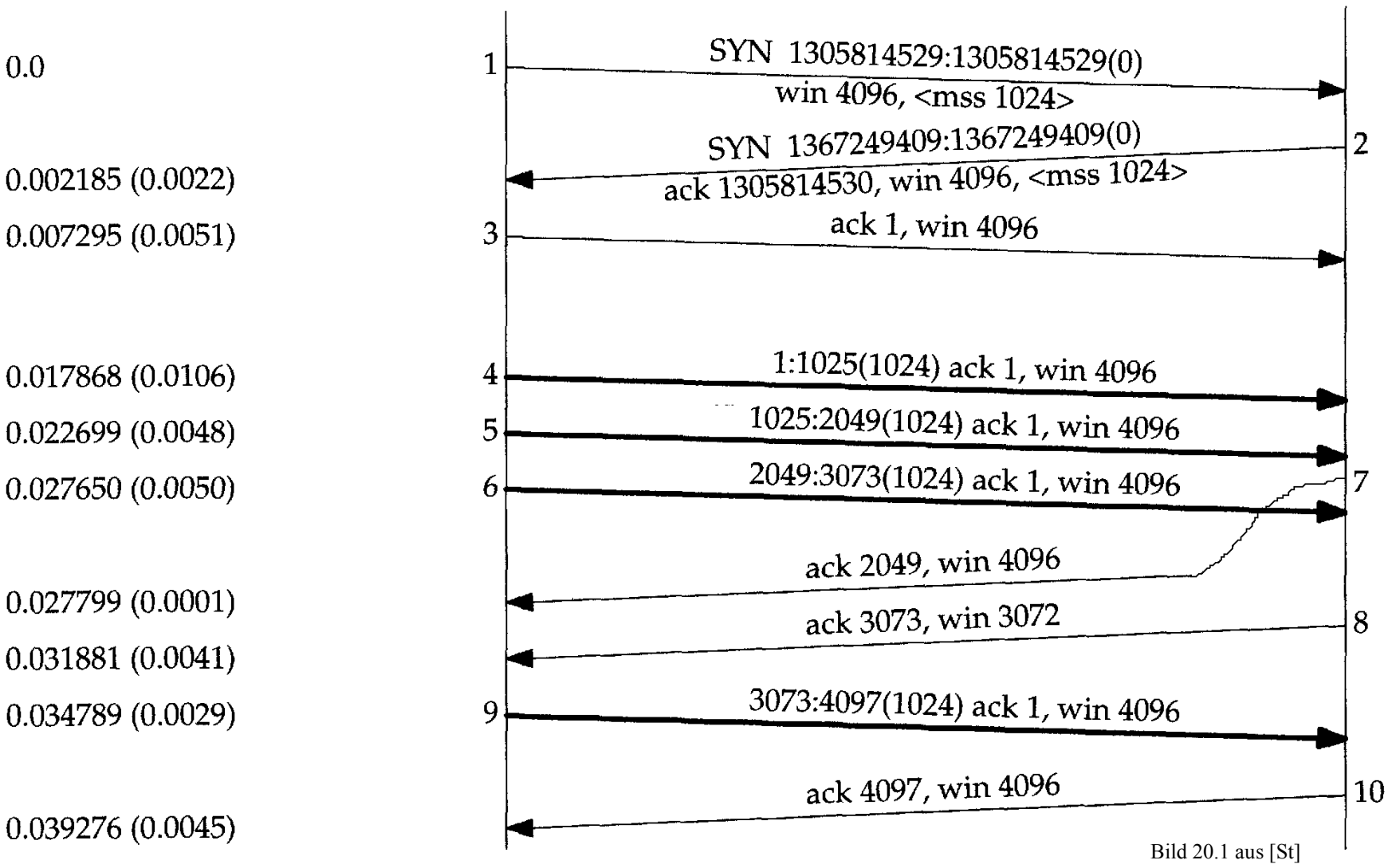



Bild 20.1 aus [St]

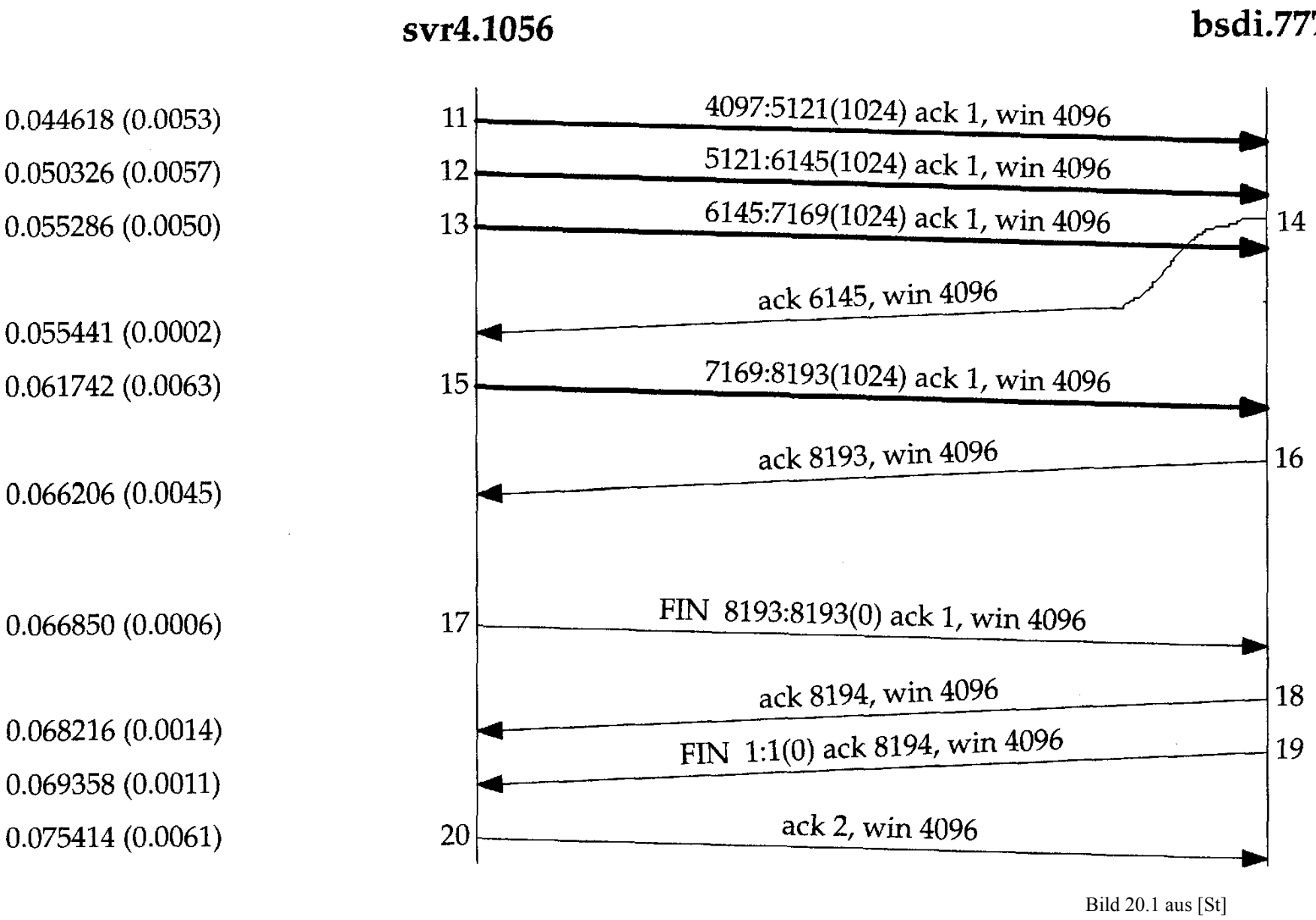



Bild 20.1 aus [St]

# Stau-Vermeidung (congestion avoidance)

- Stau (congestion) Beispiel:

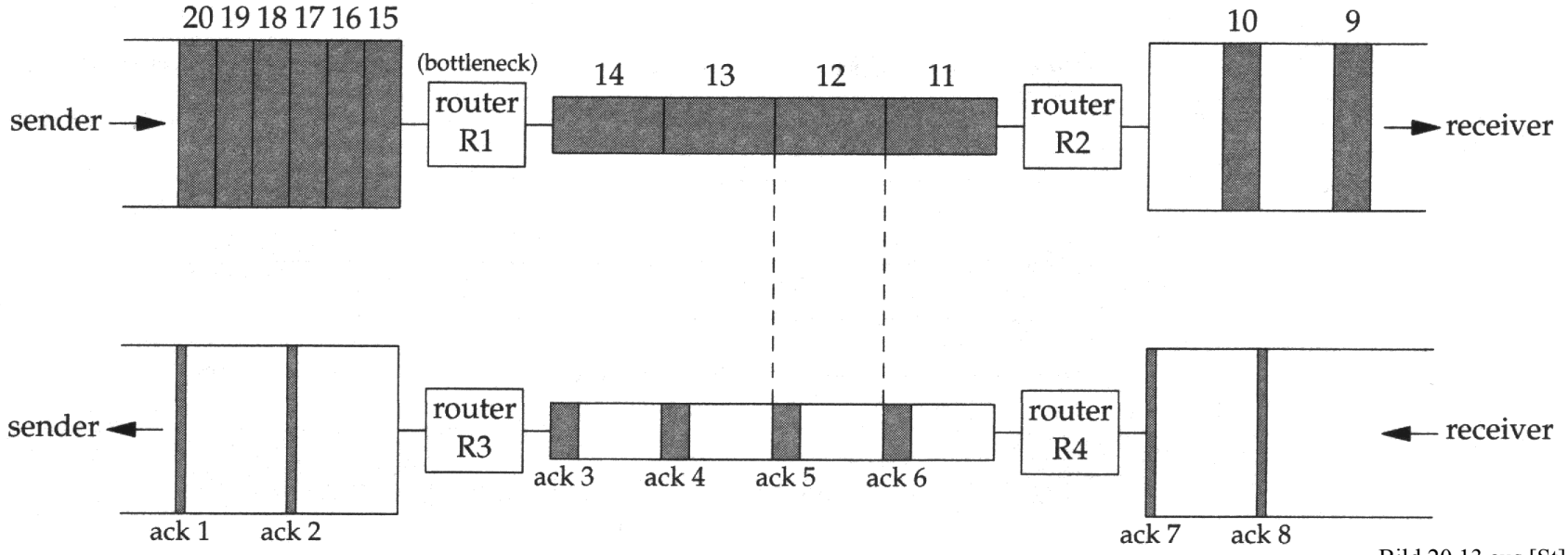



Bild 20.13 aus [St]

- Wenn R1 nicht genügend Pufferkapazität besitzt wirft er Segmente weg.

- Die Abstände der ACKs „entsprechen“ den Abständen der Datensegmentankunft.
Der Sender erkennt die Empfangsrate. (→ self-clocking)

# Stau-Vermeidung (congestion avoidance)

nach dem Prinzip: Additives Steigern, multiplikatives Senken (AIMD)

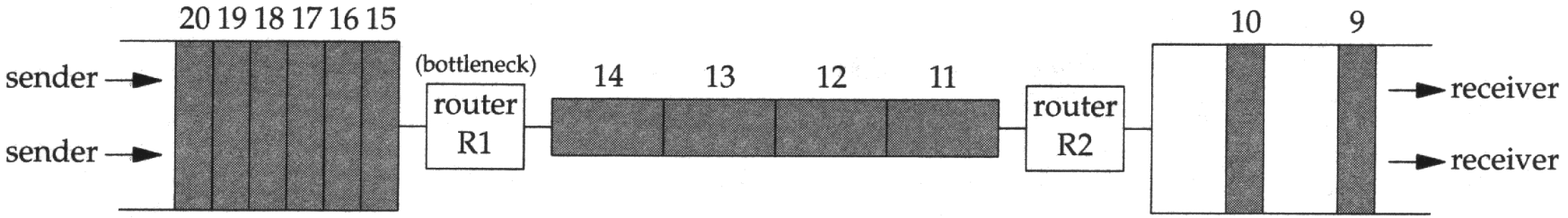
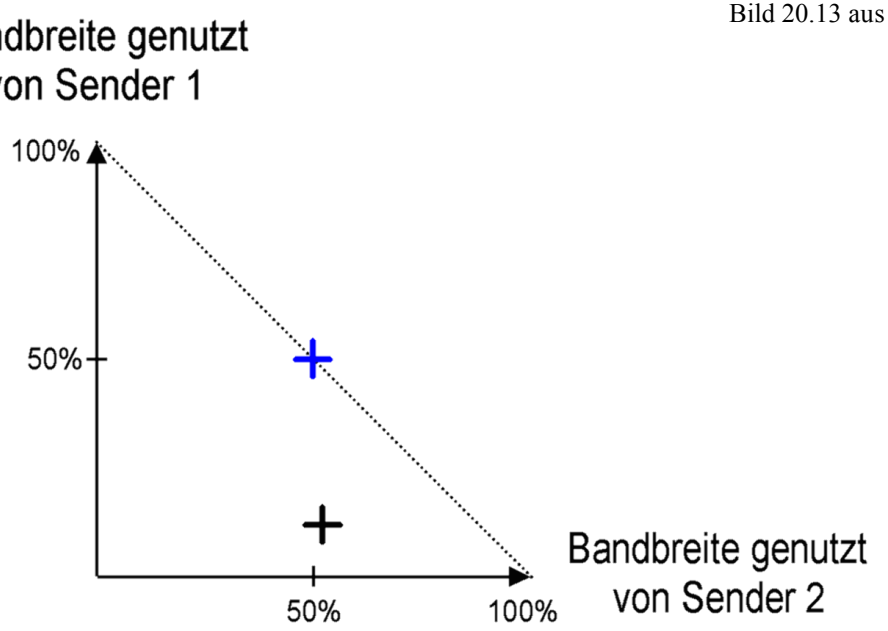



Bild 20.13 aus [St]

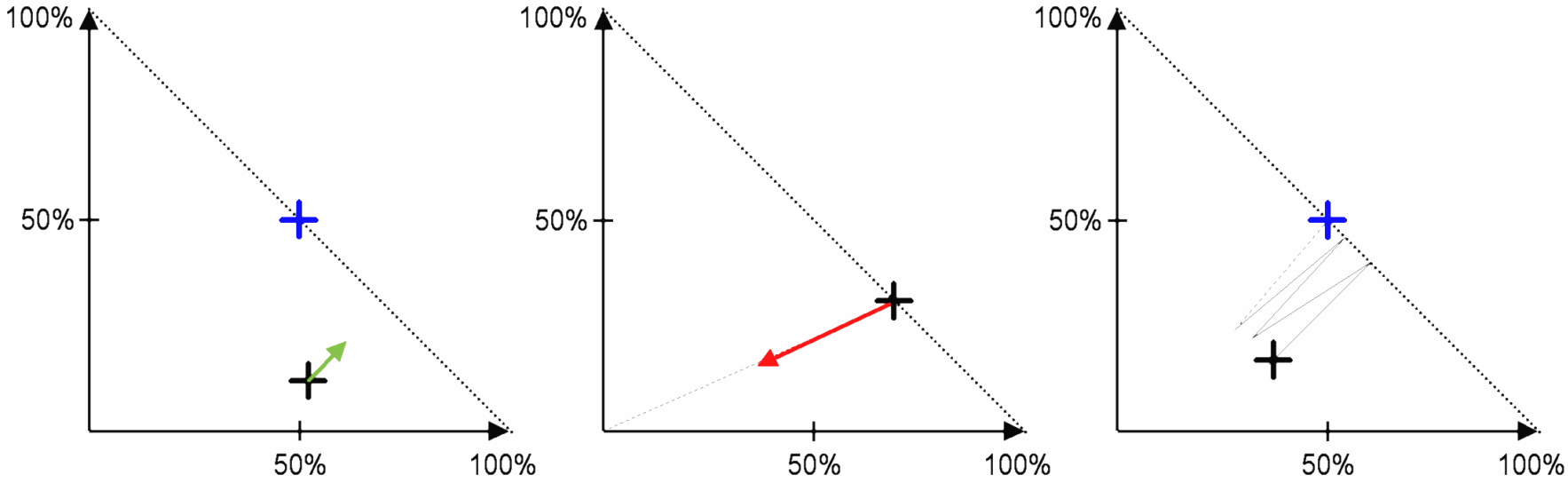
Zwei Verbindungen teilen sich eine Leitung.

**Idealzustand**:
Jede Verbindung nutzt 50% der Bandbreite.



# Stau-Vermeidung (congestion avoidance)

nach dem Prinzip: Additives Steigern, multiplikatives Senken (AIMD)




**In vielen TCP-Implementierungen:**

- Pro $RTT$ Steigerung der Senderate um 1 $MSS$ / $RTT$.
- Bei Stau: (Sofortige) Verringerung der Senderate auf die Hälfte.
- Vorgehensweise führt (theoretisch) iterativ zum Idealzustand.

# Langsamer Start (slow start)

**Nachteil des AIMD:** Die Übertragungsrate steigt nur langsam.

**Lösung: Langsamer Start**, der einen schnelleren Start bewirkt.

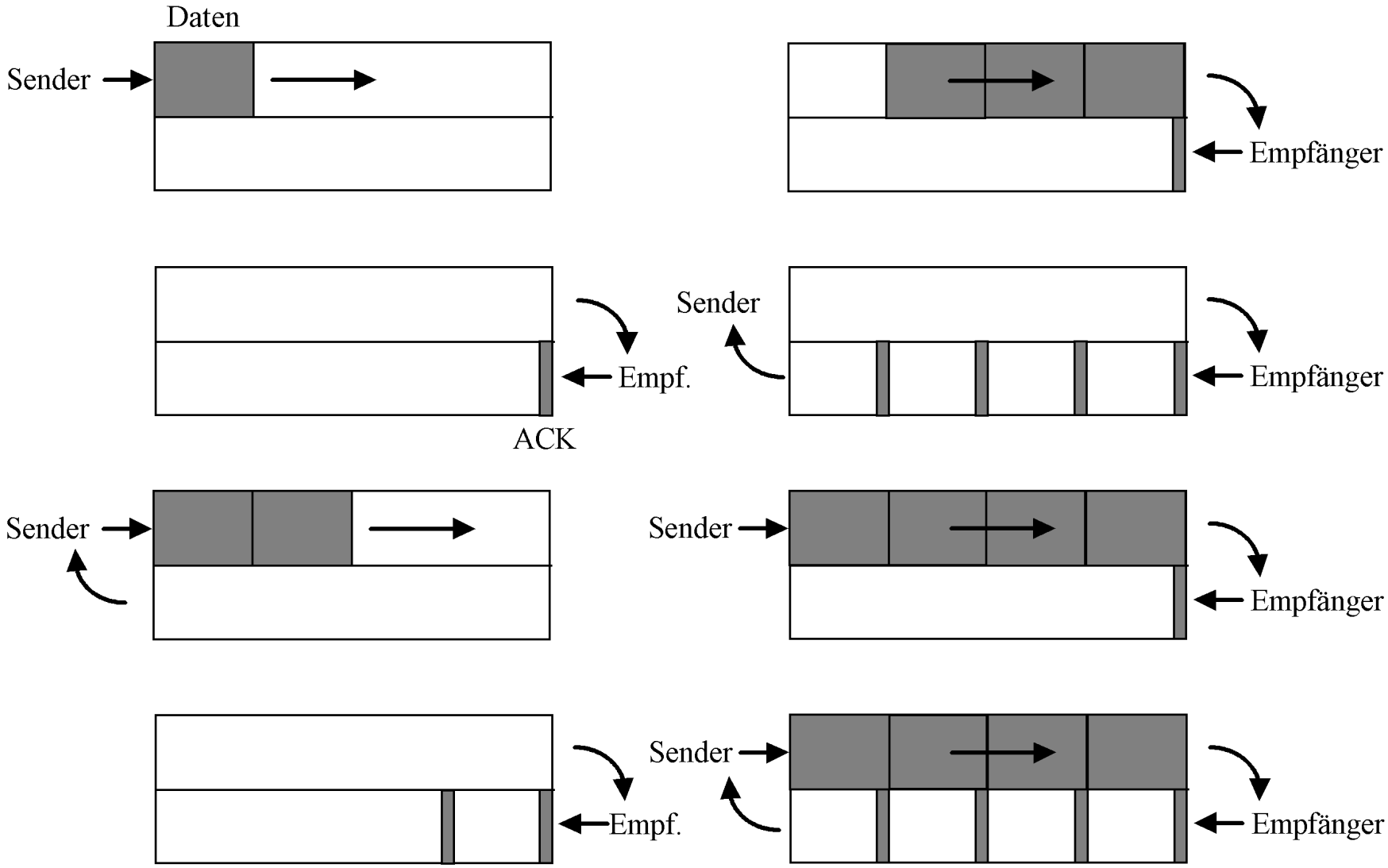
- Inzwischen von allen TCP-Implementierungen verlangt.

- Senderate wird mit 1 $MSS / RTT$ initialisiert.

- Erhält der Sender ein ACK, so erhöht er die Senderate um 1 $MSS / RTT$.

→ Exponentielles Wachstum der Senderate.

# Langsamer Start (slow start)



# Langsamer Start (slow start)

- Der letzte Zeitpunkt ist ein idealer Übertragungszustand.
  Zu jeder Zeit gleich viele Datensegmente in der Verbindung.

  ➔ Idealwert für *adwnd* ist die *Kapazität* der Verbindung.

- *Kapazität* [bits] = *Bandbreite* [bits/sec] x *RTT (round-trip time)* [sec]

- *adwnd* = *Kapazität* nicht immer möglich, da oft
  *Kapazität* größer als der Maximalwert von *adwnd* (65535 bytes)

# Zusammenspiel: S.V. und L.S.

**Wie erkennt man Stau (~ Segmentverlust)?**

2. Man erhält mehrmals (oft: 4 mal) das gleiche ACK.
   (→ Segmente werden zwar übertragen, aber es gab Verlust.)
3. Es kommt kein ACK in $RTO$ Sekunden.

**Wechsel zwischen S.V. und L.S.:**

„Langsamer Start“ bis es Stau gibt und 1. oder 2. eintritt.

# Zusammenspiel: S.V. und L.S.

**Wenn 1. oder 2. eintritt:**

- In beiden Fällen wird zunächst das verlorene Segment erneut gesendet. Und dann:

- Fall 1.: „Stau-Vermeidung“ tritt in Kraft: Die Senderate wird halbiert und „additives Steigern“ angewandt.

- Fall 2.: Hier wird die Senderate auf 1 $MSS / RTT$ zurückgesetzt und „Langsamer Start“ tritt in Aktion.
  Die Wartezeit für die nächsten ACKs ist auf $RTO$ x $2^n$, $n=1,2,...$ max 64 sec. festgelegt. (→ exponential backoff)

# TCP features und options

**presist timer:**

- Sender überprüft das *adwnd* in gewissen Zeitabständen.
- Verhindern, dass beide Seiten aufeinander warten wenn *adwnd=0* und ein ACK verloren ging.

**keepalive timer:**

- Überprüfung ob die andere Seite noch „da“ ist. (Standard: alle 2 Stunden)
- Der keepalive timer ist eine kontroverse Fähigkeit.

**urgent data:**

- Übertragung besonders wichtiger Nachrichten zwischen den „normalen“ Daten einer TCP-Verbindung.

# TCP features und options

**MTU-Ermittlung entlang eines Pfades:**

- Wird verwendet um Segmentzerstückelung zu verhindern.

**Zeitsiegel:** (timestamp)

- Genauere RTT-Messung.

**window-Skalierung:**

- *adwnd* größer als 65535 bytes.

**PAWS:** (protection against wrapped sequence numbers)

- Zusätzliche Fehlererkennung für Gigabit-Netzwerke etc.

**T/TCP:** (TCP for transactions)

- TCP Erweiterung für den schnellen Austausch weniger Segmente.

# Zusammenfassung

- TCP will Daten möglichst effizient transportieren.
- Dabei kann Überfüllung und/oder Segmentverlust auftreten.
- Dies wird mit unterschiedlichen Mitteln teilweise vermieden.
- TCP bietet rückwärtskompatible Optionen an, meist für die Verwendung mit schnellen Medien.

**Literatur:**


- [St] Stevens, W.R. 1999 „TCP/IP Illustrated, Volume 1: The Protocols“
- [Ja] Jacobson, V. 1988 „Congestion Avoidance and Control“ http://ftp.ee.lbl.gov/papers/congavoid.ps.z