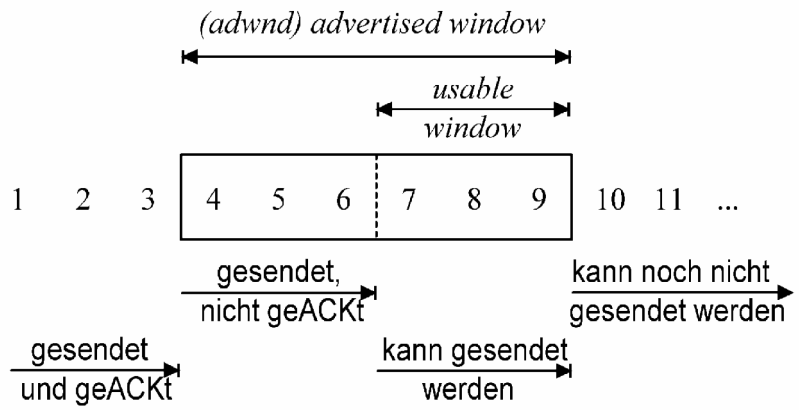
TCP Teil 2

1. Einleitung – Transport großer Datenmengen

TCP wird unter Anderem auch zum Transport von großen Datenmengen verwendet. Dieser Transport ist von TCP jedoch keineswegs einheitlich festgelegt, sondern hängt von vielen Faktoren ab. Die Schichten des Layer-Modells unterhalb TCP, hierbei besonders die Netzwerkdynamik (Kollisionen, Routing, Bandbreite,...) haben wesentlichen Einfluss auf die TCP-Performance ebenso wie die Applikationen, die TCP verwenden und damit auch die Prozessorzeitverteilung des Betriebssystems. Sehr bestimmend sind natürlich die TCP-Implementierungen selbst, die wie alles was einem ständigen Entwicklungsprozess unterworfen ist alles andere als einheitlich sind. Auf einige (bei weitem nicht alle) Konzepte, die von TCP verwendet werden soll nun im Folgenden genauer eingegangen werden.

1. Sliding Window Protocol



TCP wird auch SWP genannt, weil die Übertragung vom Empfänger durch ein Fenster (*advertised window*) gesteuert wird, das fortlaufend seine Position ändert. Die Größe entspricht normalerweise der des Empfangspuffers. Der Sender darf nicht mehr Daten senden als die Fenstergröße. Der linke Fensterrand bewegt sich vorwärts wenn der Sender ACKs (acknowledgements) bekommt. Der rechte Fensterrand wird vom Empfänger geöffnet, wenn die Applikation der Verbindung die Daten aus dem Empfangspuffer liest und damit Platz für neue Daten schafft. Die Zahlen im Bild stehen symbolisch für die übertragenen Daten und sind fortlaufend durchnummeriert (Sequenznummer).

2. Begriffe

- *MSS (maximum segment size)*

  Die *MSS* ist die max. Segmentgröße, die TCP auf einmal empfangen kann. Sie ist Hardware abhängig und wird aus der *MTU(maximum transmission unit)* abzüglich der TCP- und IP-Header berechnet.

- *RTT (round-trip time)*

  Die *RTT* gibt das Zeitintervall an, das eine Übertragung eines Datensegmentes vom Sender zum Empfänger zusammen mit dem Rücktransport des Zugehörigen ACKs in Anspruch nimmt.
  Der Wert der *RTT* wird von TCP für jede Verbindung gemessen, dabei wird immer nur eine Messung gleichzeitig vorgenommen und Messergebnisse von wiederholt versendeten Daten ignoriert. Neuere TCP-Implementierungen verwenden zur genaueren Messung die Timestamp-Option.

- *RTO (retransmission timeout)*

  Hierbei handelt es sich um einen Zeitwert den TCP auf ein ACK vom Empfänger wartet bevor es annimmt, das die gesendeten Daten verloren gegangen sind und diese erneut sendet. Der Wert des *RTO* wird laufend neu berechnet aus einem gedämpften *RTT*-Schätzer und der mittleren Abweichung der *RTT*-Werte.

2. Stau-Vermeidung (congestion avoidance)

Was ist Stau?

Stau entsteht, wenn über eine Leitung mehr Daten geschickt werden sollen, als die Bandbreite der Leitung erlaubt. Dies macht sich bei Routern bemerkbar, die an Verbindungen verschiedener Bandbreite arbeiten. Kommen auf der einen Seite mehr Daten an als auf der anderen abfließen, füllt sich der Eingangspuffer des Routers. Ist dieser voll, so bleibt dem Router keine andere Wahl als Datensegmente wegzuwerfen.

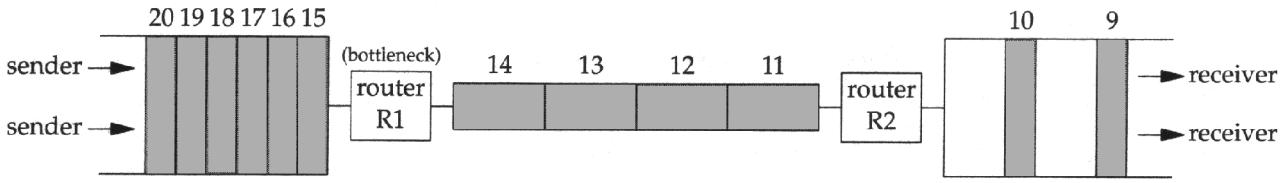
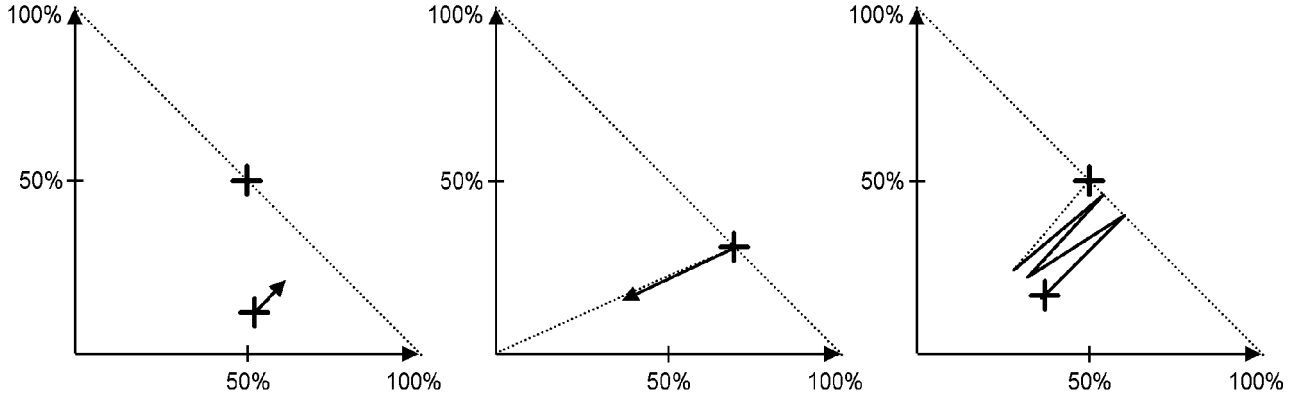


Bild 20.13 aus [St]

Wie wird Stau vermieden?

Dazu kommt das Prinzip „Additives Steigern, multiplikatives Senken“ (additive increase, multiplikative decrease) zum Einsatz, das auch eine gleichmäßige Verteilung der Bandbreite unter konkurrierenden Sendern bewirkt.



Zwei TCP-Verbindungen senden über eine Leitung:
x-Achse: Leitungsbandbreite genutzt von Sender 1
y-Achse: Leitungsbandbreite genutzt von Sender 2

Um die zur Verfügung stehende Bandbreite zu jeder Zeit auszunutzen versucht jeder Sender seine Senderate zu erhöhen. Dies geschieht linear (additives Steigern) um 1 *MSS* pro RTT^2 und schlägt sich im Graphen in einer Bewegung in Richtung der 1.Winkelhalbierenden nieder (Bild 1)

Dies geht gut bis die Leitung auf voll belastet ist. Beim Steigern über diese Grenze hinaus tritt Segmentverlust ein. Jetzt senkt jeder Sender seine Rate auf die Hälfte (multiplikatives Senken). Die resultierende Bewegung im Graphen ist zum Ursprung hin gerichtet. (Bild 2)

Dieses Vorgehen wird in ständiger Wiederholung fortgesetzt und führt iterativ zu dem Zustand, bei dem beide Sender jeweils die Hälfte der Bandbreite nutzen. Obwohl sich dieses Modell ohne Probleme auf mehr als zwei Verbindungen verallgemeinern läßt, gibt es so einen Idealzustand in der Praxis nicht, da laufend Verbindungen auf- und abgebaut werden und die Verbindungen auch nicht zu jeder Zeit Daten senden wollen.

3. Langsamer Start (slow start)

Zusätzlich zur Stauvermeidung möchte man jedoch, dass ein hoher Datendurchsatz möglichst schnell erreicht wird, was im Gegensatz zum additiven Steigern des obigen Verfahrens steht. Um diesem Ziel näher zu kommen wird laut RFC von allen neuen TCP-Implementierungen ein Algorithmus namens „slow start“ verlangt, der einen schnelleren Anstieg der Senderate gewährleistet.

Dabei wird mit einer Senderate von 1 *MSS* pro *RTT* begonnen und die Rate für jedes geACKte Segment um 1 *MSS* pro *RTT* erhöht. D.h. man sendet zunächst ein Segment, erhält nach 1 *RTT* das ACK, erhöht und sendet dann also zwei Segmente, erhält nach 1 *RTT* die beiden ACKs, erhöht für jedes Segment und sendet vier Segmente (Bild)...

Diese Vorgehensweise bewirkt letztendlich einen exponentiellen Anstieg der Senderate.

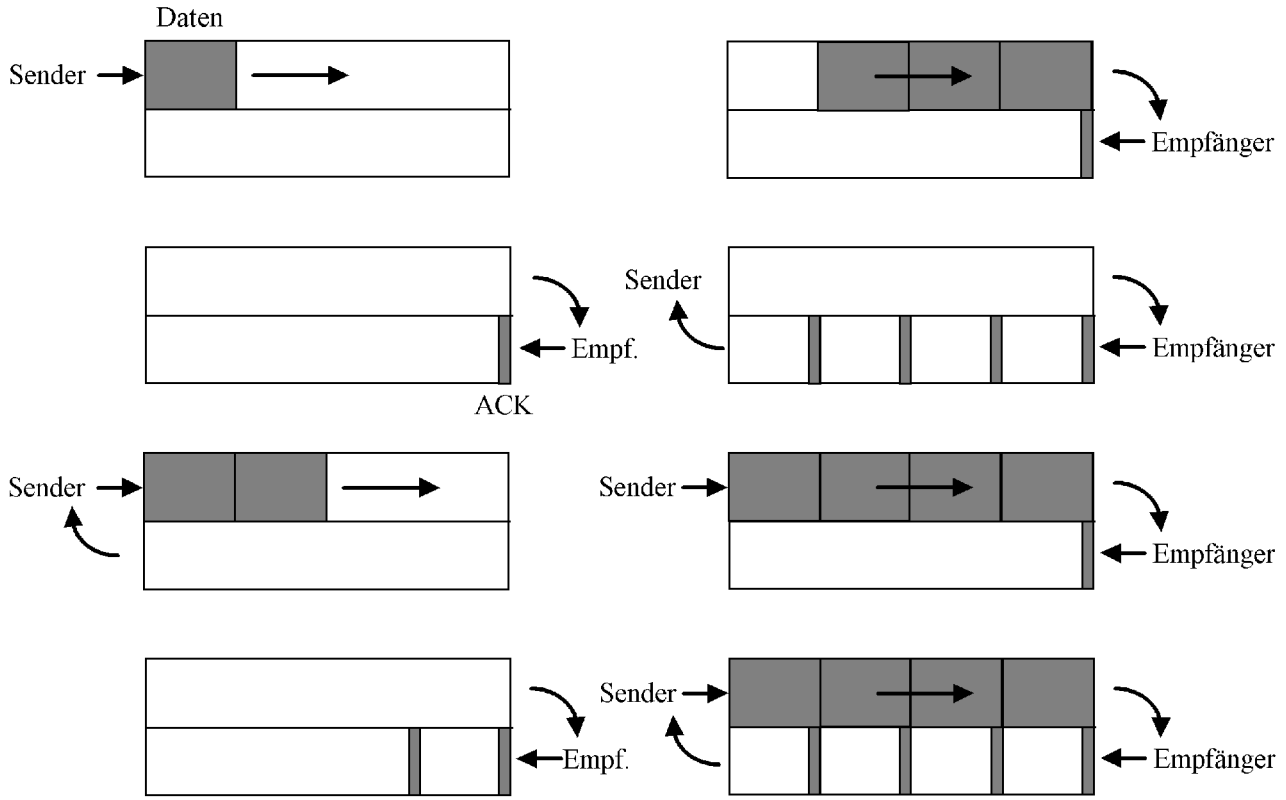
Wie man dem Bild entnehmen kann haben die zurückkommenden ACKs die im Vergleich zu den Datensegmenten viel kürzer sind den gleichen Abstand wie die beim Empfänger ankommenden Segmente. Der Sender kann also die Ankunftsrate beim Empfänger in gewissen Grenzen erkennen. Dies nennt man „selfclocking“. Verfälscht werden die Abstände bei der Rückleitung der ACKs wiederum durch das Netzwerk. Die ACKs können z.B. eine ganz andere Route nehmen als die Datensegmente.

Ebenfalls auffällig am Bild ist die Tatsache, dass die Bandbreite der Leitung nur ausgenutzt werden kann wenn genügend Segmente im Umlauf sind (im Bild: acht). Um dies überhaupt erst möglich zu machen

muss allerdings das *advertised window* des Empfängers groß genug sein. Man erhält also als Idealwert für *adwnd* gerade die *Kapazität* der Leitung.

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

Dieser günstige Wert kann allerdings oft nicht erreicht werden, da er ja durch die Puffergröße des Empfängers und durch den Maximalwert von 65535 bytes (kann durch die Windowskalierung Option verbessert werden) eingeschränkt ist.



3. Zusammenspiel: „Stau-Vermeidung“ und „Langsamer Start“

Was nun nötig wird ist eine Integration dieser beiden konträren Algorithmen. Man möchte also schnell loslegen und wenn es zu Stau kommt diesen wenn möglich zukünftig vermeiden. Als erste Frage stellt sich daher: **Wie erkennt man Stau?**

Stau erkennt man daran, dass Datensegmente nicht beim Empfänger ankommen, sei es weil sie von einem Router weggeschmissen wurden weil sein Puffer voll ist oder weil ihre Lebenszeit (time-to-live) abgelaufen war. Der Sender bemerkt dies in der Praxis dadurch, dass er kein ACK für das entsprechende Segment bekommt. Hierbei fließt die Tatsache mit ein, dass auf den meisten Medien der Segmentverlust durch Übertragungsfehler sehr gering ist. Daraus folgen zwei Szenarien in denen der Sender davon ausgeht, dass Stau vorliegt:

1.) Der Sender bekommt mehrmals (meist vier mal) das gleiche ACK. Daraus läßt sich schließen, dass zwar die Datenübertragung nicht unterbrochen ist, aber dass das unmittelbar auf das mehrmals geACKte Segment folgende verloren gegangen ist.

2.) Der Sender bekommt kein ACK für ein versendetes Datensegment innerhalb des Zeitintervalls *RTO*

Wie wird das Zusammenspiel von S.V und L.S. realisiert?

Der Sender beginnt zu senden mittels der „Langsamer Start“-Technik, was früher oder später, wenn die Leitung voll belastet ist, zu Stau führt. Dann wird das verlorene Segment erneut gesendet und im Fall

1.) geht man zur Stau-Vermeidung über. D.h. die Senderate wird halbiert und im Weiteren additives Steigern wie oben beschrieben angewandt.

2.) wird die Senderate auf 1 *MSS* pro *RTT* zurückgesetzt und „Langsamer Start“ tritt in Kraft. D.h. der Sender wartet jetzt erst mal wieder auf das ACK für das erneut versendete Datensegment. Das Zeitintervall berechnet sich für diesen Fall nach $t = RTO \times 2^n$ mit der Nummer des Übertragungsversuches $n=1,2,3,...$ Man verwendet also exponentiell wachsende Wartezeiten mit einem Maximum von 64 sec. (exponential backoff)

4. TCP features und options

- **presist timer**

  Veranlasst, dass der Sender von Zeit zu Zeit eine Überprüfung des *adwnd* vornimmt, um zu verhindern, dass beide Seiten aufeinander warten wenn *adwnd=0* und das letzte ACK verloren ging. Dies geschieht durch das Senden eines sogenannten Fenstertests (window-probe) auf den der Empfänger mit einem ACK, das natürlich auch die aktuelle Fenstergröße enthält, antworten muss.

- **keepalive timer**

  Wird von einer Verbindung, bei der längere Zeit kein Segment übertragen wird, dazu verwendet zu prüfen ob die andere Seite noch „da“ ist. Das Zeitintervall ist einstellbar und verwendet einen Standard von zwei Stunden. Dieses Feature ist kontrovers diskutiert. Viele sind der Meinung, dass es nicht in die Transport- sondern in die Anwendungsschicht gehört. Des halb darf es auch nur auf ausdrücklichen Wunsch der Applikation aktiviert werden.

- **urgent data**

  Bietet eine Möglichkeit bestimmte Bytebereiche innerhalb der laufenden Übertragung als besonders wichtige Daten zu kennzeichnen.

- **MTU-Ermittlung entlang eines Pfades (path MTU discovery)**

  Man möchte möglichst vermeiden, dass Segmente beim Versand zerstückelt werden. Dazu muss man aber die maximale Segmentgröße wissen, die über einen Pfad versendet werden kann. Dies ist natürlich die „kleinste“ aller betroffenen Leitungen. Diese Option ermittelt allerdings die *MTU*. Die maximale Segmentgröße ergibt sich dann durch Subtraktion der TCP- und IP-Header

- **window-Skalierung (window scale option)**

  Mit dieser Option ist es möglich die Grenzen der adwnd zu durchbrechen und Fenstergrößen bis zu 1 GByte zu verwenden. Dies wird erreicht indem alle Fenstergrößen der Header mit einem beim Verbindungsaufbau festgelegten Faktor multipliziert werden. Man braucht dies hauptsächlich für Netze mit hohen Übertragungsraten, die folglich eine hohe Leitungskapazität haben.

- **Zeitsiegel (timestamp option)**

  Ist diese Option aktiviert, so wird jedes gesendete Segment mit einem Zeitwert versehen. Diese Werte werden dann vom Empfänger zusammen mit den zugehörigen ACKs zurückgespiegelt und ermöglichen so eine sehr genaue Messung der *RTT* der Verbindung.

- **PAWS (protection against wrapped sequenze numbers)**

  Bei Netzen sehr großer Kapazität (Gigabit-Netzwerke) kann es vorkommen, dass die fortlaufende Sequenznummerierung im TCP-Header in kurzer Zeit einen Überlauf hat und so mehrere Segmente mit der gleichen Sequenznummer unterwegs sind. Für dennoch eine richtige Einordnung dieser Segmente sorgt die PAWS Option, wozu sie die Zeitsiegel verwendet.

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

  Diese Erweiterung, die kaum verwendet wird, verkürzt den Verbindungsauf- und Abbau in Fällen in denen nur schnell wenige Segmente (Nachrichten) ausgetauscht werden sollen.

4. Zusammenfassung und Literatur

TCP bietet für Anwendungen einen verläßlichen Datenstrom über das IP an. Dabei versucht es die Daten auf Leitungen, die von mehreren Verbindungen beansprucht werden, möglichst effektiv und effizient zu transportieren und die entstehenden Konkurrenzen auszugleichen. Ein Problem das dabei auftreten kann ist Stau. TCP versucht diesen mit geeigneten Algorithmen so gering wie möglich zu halten und dennoch einen maximalen Datendurchsatz zu erreichen.

Des Weiteren bietet TCP einige Optionen die durch ihre rückwärtskompatible Implementierung die Fähigkeiten von TCP ausbauen und auch seine Einsatzmöglichkeit auf neuen Netzen mit hohen Bandbreiten gewährleisten. Optionen können für eine Verbindung allerdings nur dann zum Einsatz kommen wenn sie auf beiden Seiten implementiert sind.

Verwendete 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