Seminararbeit

the protocols
that run the internet

Internet Networking – TCP Congestion Avoidance and Control

Philipp Liebrenz

Universität Konstanz
FB Informatik und Informationswissenschaft

INHALT

1 EINLEITUNG

Das Internet hat heute über 600 Millionen Nutzer[1]. Und täglich kommen mehrere Tausend hinzu. Der Boom des Internets begann Ende der 80er Jahre. Immer mehr Organisationen und Universitäten schlossen sich dem bis dahin nur kleinen Netzwerk an. Während es 1987 weltweit nur 10 000 Server gab, waren es 1989 bereits über 100 000 Server[2]. Und der Zuwachs wurde immer stärker. Doch die Knotenpunkte des jungen Netzes waren mit diesen Massen überfordert.
Immer öfter kam es zur Überlastung (engl. Congestion) des Netzes. Die Router, die für das Weiterleiten der Datenpakete zuständig waren, konnten die Anfragen nicht mehr bewältigen. Es drohte der völlige Zusammenbruch des Netzes. Neue Mechanismen zur Kontrolle der Überlastung (engl. congestion control) und der Überlastvermeidung (engl. congestion avoidance) mussten geschaffen werden. Maßgeblich beteiligt an der Entwicklung der noch heute benutzen Methoden waren Van Jacobson, damals Lawrence Berkley Laboratory und Michael J. Karens, damals University of California at Berkeley.

Diese Seminararbeit beschreibt die Mechanismen und deren Implementierungen, die in dem Paper „Congestion Avoidance and Control“ [8] von Jacobson und Karels vorgestellt wurden und die Modifikationen bis hin zu den heutigen Standards innerhalb des TCP Protokolls.

[1] http://www.nua.com/, Stand: September 2002
[2] http://www.kbs-koeln.de/vonbodel/kif/grundlag.htm, 22. Juni 2003

2 GRUNDLAGEN

Um zu verstehen wie und wo die Methoden des Congestion Avoidance ansetzen, ist es unumgänglich einige Grundlagen der Netzwerkkommunikation zu betrachten. Insbesondere spielt das TCP- Protokoll eine entscheidende Rolle bei derUmsetzung der Überlastkontrolle. In diesem Kapitel wird das TCP-Protokoll und dessen Rolle in der Netzwerk-kommunikation vorgestellt.

2.1 Netzwerkkommunikation

Ein Kommunikationssystem aus der Sicht von TCP lässt sich in vier Schichten unterteilen: In die Anwendungsschicht, die Host-zu-Host Transport Schicht, die Internet-Schicht und die Netzwerk-Interface-Schicht. Jede dieser Schichten ist unabhängig und kommuniziert nur mit der darüberliegenden und der darunterliegenden Schicht.

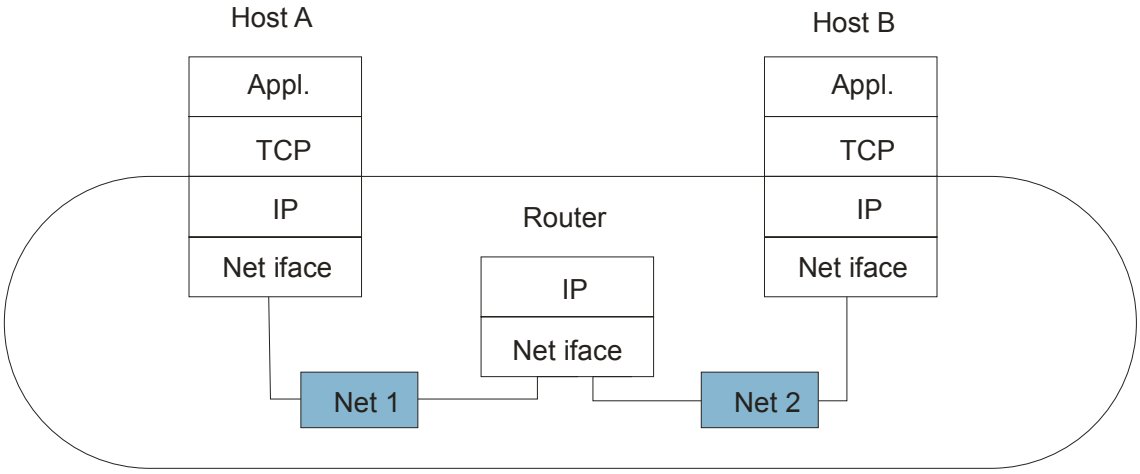

Abbildung 1: Netzwerkkommunikation aus der Sicht von TCP

In der untersten Schicht, der **Netzwerkschicht,** werden die Datenpakete aus der IP-Schicht auf das Netzwerkkabel gegeben. Doch zuvor werden die Pakete mit einem Header versehen, das heißt, an die eigentlichen Daten wird eine zusätzliche Bit-Folge gehängt. In dieser Bit-Folge werden u. a. die Hardware-Adresse des Empfängers sowie eine Prüfsumme der Daten gespeichert. Erst dann wird das Paket zugestellt. Der Empfänger überprüft die Prüfsumme und vergleicht die eigene Hardware-Adresse mit der Empfängeradresse des Headers und leitet das Paket bei Übereinstimmung an die höheren Schichten weiter.

In der **Internetschicht** befindet sich das IP-Protokoll. Es ist für das Routing zuständig und damit auf welchem Weg die Datenpakete an ihr Ziel gelangen. Auch hier wird das Datenpaket mit einen Header versehen. In ihm werden diesmal u.a. die Quell- und die Ziel-IP-Adresse gespeichert. Mit der IP-Adresse können Rechner in einem Netzwerk adressiert werden.

Befindet sich die Ziel-IP-Adresse im gleichen Teilnetzwerk (Subnet), kann das Paket direkt zugestellt werden. Ansonsten wird das Paket mit Hilfe einer Routing-Tabelle weitergeleitet bis es im richtigen Subnet ist und zugestellt werden kann. Findet das Paket innerhalb einer gewissen Anzahl an Routern nicht zum Zielhost, wird es gelöscht.

Durch das IP-Protokoll ist also nicht gewährleistet, ob das Paket überhaupt sein Ziel erreicht. Auch können Daten-Pakete beim Transport stark verzögert oder sogar dupliziert werden. Für einen sinnvollen Datenaustausch zweier Hosts bzw. Anwendungen sollte aber die Zuverlässigkeit der Zustellung gegeben sein.

Das TCP der **Transportschicht** ist für die Sicherstellung dieser Zuverlässigkeit zuständig. Es beschreibt wie eine Verbindung zwischen den Hosts hergestellt wird, wie Fehler erkannt werden, und wie Datenpakete prompt und zuverlässig zugestellt werden, ohne dabei das Netz zu überlasten.

2.2 TCP - Transport Control Protocol

Aus der Applikationsschicht bekommt TCP die Daten, die übermittelt werden sollen in Form eines Bitstroms. TCP teilt diese in höchsten 64 KByte große Segmente auf. Diese Segmente werden später als IP-Pakete (IP-Datengramme) verschickt. Da das IP-Protokoll auch nicht sicherstellen kann, dass die Datengramme in der richtigen Reihenfolge zugestellt werden, muss auch hierfür TCP Sorge tragen. TCP versieht die Pakete mit dem TCP-Header. Für jedes Paket wird dort eine eindeutige, fortlaufende Sequenznummer gespeichert. Dadurch ist eine eindeutige Reihenfolge definiert und der Bit-Strom lässt sich nach der Übermittlung rekonstruieren.

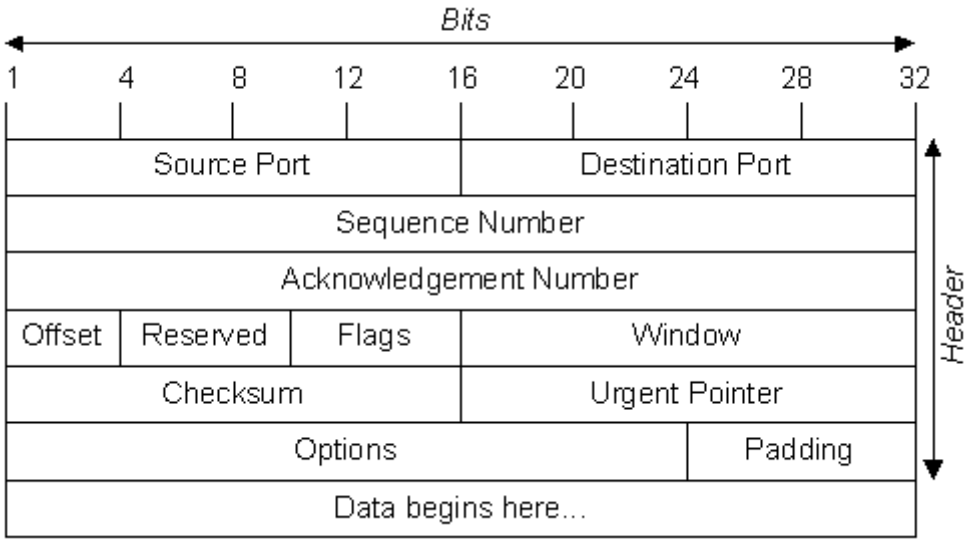

Abbildung 2: Der TCP-Header

Bevor der Datenaustausch beginnen kann, wird zwischen dem Sender und dem Empfänger eine „virtuelle Verbindung“ aufgebaut. Dies geschieht über einen sogenannten „*3-Wege-Handschlag*“. Der Sender bittet um eine Verbindung, der Empfänger stimmt zu und der Sender bestätigt den korrekten Verbindungsaufbau. Die Verbindung wird als virtuell

bezeichnet, da eine feste, zuverlässige Verbindung aufgebaut wird, obwohl sie auf einer verbindungslosen Übertragung beruht. Nun kann mit der Übertragung begonnen werden.

2.3 Konzepte des zuverlässigen Datenaustauschs

TCP muss sicherstellen, dass ein Datenpaket auch wirklich beim Empfänger ankommt. Die grundlegendste Technik dies zu gewährleisten ist, dass der Empfänger für jedes empfangene Datenpaket eine Empfangsbestätigung (Acknowlegement oder kurz: ACK) an den Sender zurückschickt. Erst wenn der Sender diese Bestätigung erhält, kann er sicher sein, dass das Datenpaket angekommen ist. Dieses Konzept nennt man „*Bestätigtes Senden*“.

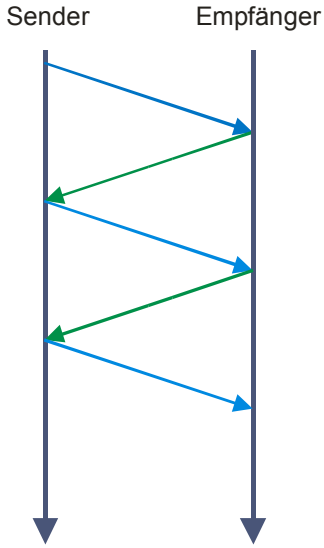

Abbildung 3: Bestätigtes Senden

Trifft die Bestätigung nach einer geschätzten Zeit nicht ein, muss davon ausgegangen werden, dass das Paket oder die Bestätigung auf dem Weg verloren gegangen ist und das Paket wird noch einmal geschickt („*Bestätigtes Senden mit Neuübertragung*“).

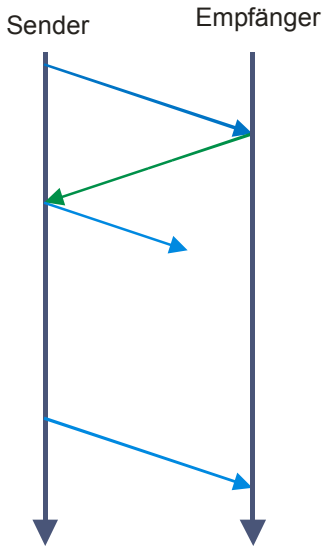

Abbildung 4: Bestätigtes Senden mit Neuübertragung

Zusammen mit der fortlaufenden Nummerierung der Pakete ist das „Bestätigte Senden mit Neuübertragung“ ein effektives Konzept des zuverlässigen Datenaustausches. Allerdings ist es nicht effizient. Bei hohen Latenzzeiten ist der Sender sehr lange untätig, da er immer auf die Bestätigung des letzten Datensegments warten muss.
Um die Effizenz zu steigern, bedient man sich des „*Sliding Window*“- Konzepts. Statt auf die Bestätigung jedes einzelnen Paketes zu warten, erlaubt man, eine durch ein *Sendefenster* definierte Anzahl von Paketen auch unbestätigt zu senden. Das Fenster gibt den Rahmen vor, welche Pakete gerade gesendet werden dürfen. Wird nun das erste Paket der bereits gesendeten Segmente bestätigt, verschiebt sich das Fenster um ein Datenpaket nach rechts (siehe Abbildung) und ein weiteres Paket tritt in den Rahmen und kann gesendet werden. Alle Pakete, die sich links des Fensters befinden, sind also versendet und bestätigt. Pakete innerhalb des Fensters werden gesendet und warten auf Bestätigung. Alle Paket rechts des Fensters sind noch nicht gesendet.

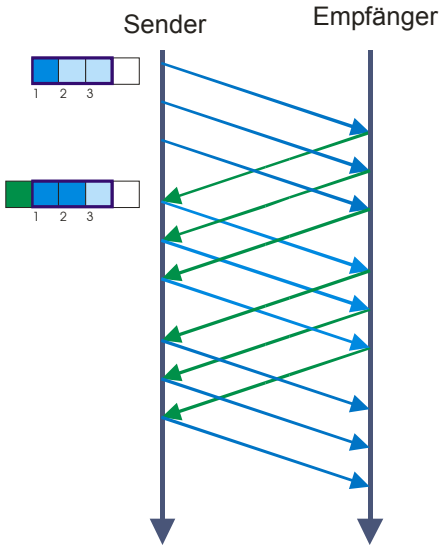

Abbildung 5: Das "Sliding Window"-Konzept

Auf der Seite des Empfängers gibt es ebenso ein Fenster, in das die empfangenen Pakete eingeordnet werden. Durch die Sequenznummern der Pakete kann bestimmt werden, welche Pakete innerhalb des Fensters noch ausstehen. Mit der Bestätigung teilt der Empfänger auch mit, welches Paket als nächstes erwartet wird. Diese Information wird im Acknowledgement gespeichert. Dadurch können mehrere bereits empfangene Pakete auf einmal bestätigt werden. Dies ist ein weiterer Effizenzvorteil, da weniger Bestätigungen versendet werden müssen.

TCP setzt das Konzept des Sliding Window um und sichert somit den effektiven und effizenten Datenaustausch zwischen Sender und Empfänger.

Zusammengefasst lässt sich sich über die Eigenschaften von TCP sagen: TCP ist

- **verbindungsorientiert**
  Zwischen Sender und Empfänger wird eine virtuelle Verbindung aufgebaut
- **eine Streaming-Schnittstelle**
  TCP erhält die Daten als Bitstom und zerlegt sie in einzelne Segmente zum Datenaustausch
- **zuverlässig**
  Durch das Konzept des „Bestätigten Sendens mit Neuübertragung“ stellt TCP sicher, dass die Daten auch wirklichen den Empfänger erreichen

Eine weitere Eigenschaft von TCP ist, dass es Funktionen der Überlastkontrolle übernimmt.

Im Folgenden wird diese Eigenschaft von TCP näher vorgestellt und die Art und Weise, wie TCP die Überlastkontrolle umsetzt, näher erläutert.

3 Grundlagen und Konzepte des Congestion Controls

3.1 Was ist Congestion control?

Wie Eingangs erwähnt, entsteht Überlastung oder „Congestion“ immer dann, wenn das Netzwerk bzw. ein Router durch zu viele Datenpakete überfordert wird. Die Übermittlung der Pakete bleibt dann aus und der zuverlässige Datenaustausch ist gestört. Die Ursachen hierfür sind simpel: Wenn mehrere Sender mit voller Kapazität Daten senden, aber es innerhalb des Netzwerkes Engstellen, so genannte „*bottlenecks*“ gibt, kann ein Router die Daten nicht so schnell übermitteln wie neue Daten nachkommen. Der Puffer des Routers läuft über und Daten gehen verloren. Verschärfend kommt hinzu, dass wenn der Sender keine Bestätigung von Empfänger erhält, er die Daten ein weiteres Mal sendet, was wiederum das Netz belastet.

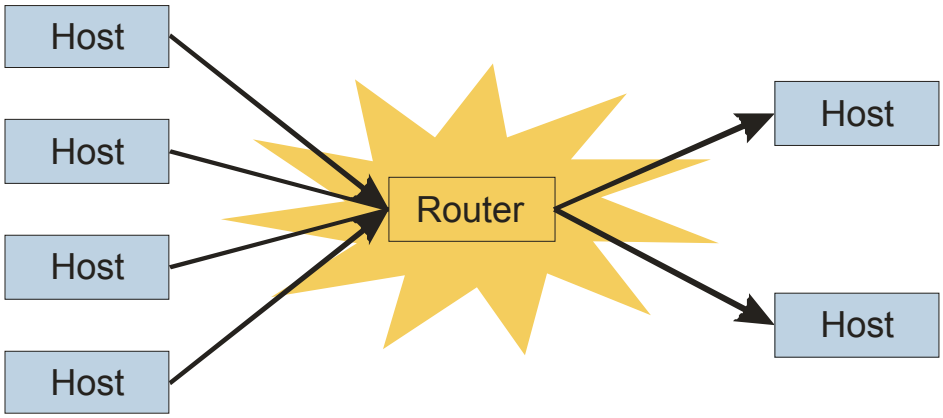

Abbildung 6: Beispiel eines überlasteten Routers

Die in TCP integrierte Überlastkontrolle (engl. „*Congestion Control*“) versucht diesen Zustand der Überlastung des Netzes zu verhindern und auf Anzeichen für Überlastung zu reagieren. Gleichzeitig sollen die verfügbaren Ressourcen effektive genutzt werden und gerecht verteilt werden.

3.2 Self-Clocking Prinzip

Durch das Konzept des „Sliding Window“ stellt TCP sicher, dass ein Datenpaket seinen Empfänger effizient erreicht. Neue Pakete werden nur dann gesendet, wenn der Sender eine Empfangsbestätigung erhält. Oder anders formuliert: Erst wenn ein altes Paket das Netzwerk verlässt, wird ein neues Paket auf das Netzwerk gegeben. Neben der Zuverlässigkeit der Zustellung wird dadurch auch erreicht, dass das Netz nicht durch zu viele Pakete überlastet wird. Man bezeichnet dies als „*conservation of packets*“. Ein solches System befindet sich in einem Gleichgewicht (engl. Equilibrium), das in Bezug auf Überlastung stabil sein sollte.

Das Schaubild illustriert ein solches Equilibrium:

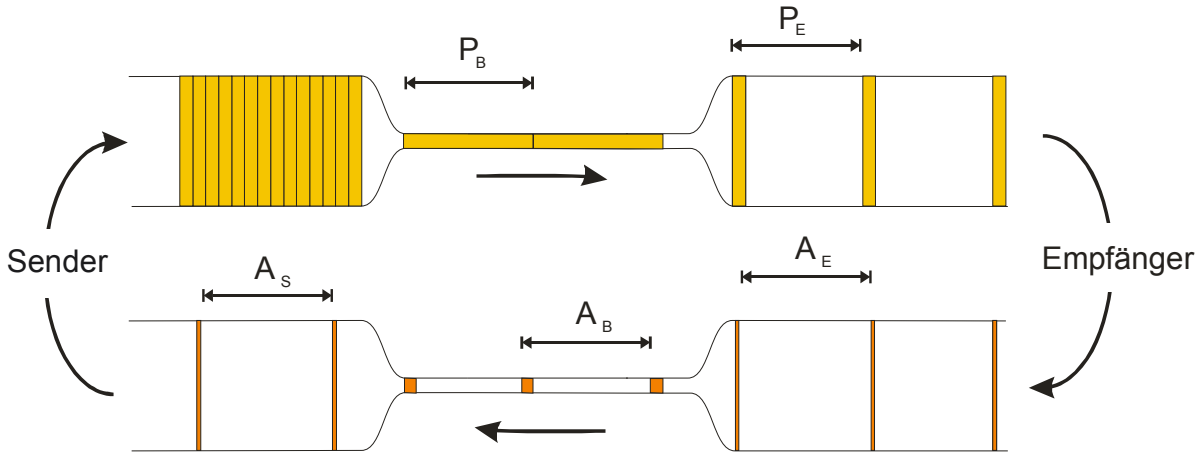

Abbildung 7: Das "Self-Clocking" Prinzip

Sender und Empfänger sind über ein Netzwerk verbunden. Ein Teilabschnitt der Verbindung ist eine langsamere Verbindung. Die Vertikale stellt die Bandbreite der Verbindung dar. Die gelben Elemente sind Datenpakete, die versendet wurden. Die Horizontale stellt die Zeit dar. Die Datenpakete brauchen länger zum Durchqueren des langsameren Abschnitts. Sie werden zeitlich gedehnt. Beim Eintritt in ein schnelleres Netz können sie wieder schneller übertragen werden. Allerdings bleibt der Abstand zwischen den Paketen bestehen (P_B=P_E). Der Empfänger bestätigt den Erhalt eines Pakets mit einem ACK (rot). Auch hier bleibt der Abstand zwischen den Paketen erhalten (A_E=A_B=A_S=P_B=P_E). Der Sender kann nun für jedes erhaltene ACK ein weiteres Paket versenden. Das System „taktet“ sich also selbst und

richtet sich nach dem langsamsten Abschnitt der Übermittlung. Es wird daher auch „*self-clocking*“ genannt.

Eine zentrale Aufgabenstellung der Überlastkontrolle ist es das Eqilibrium sicherzustellen und dadurch Überlastung zu verhindern.

3.3 Probleme bei der Sicherstellung des Eqilibriums

Befindet sich ein System im Equilibrium, sollte es nicht zu Überlastung kommen. Allerdings hat sich am Beispiel des Internets gezeigt, dass es Probleme beim Erreichen dieses Zustands gibt. Im Besonderen gibt es drei Arten:

Problem 1: Das System kommt nicht in den Zustand des Equilibriums.

→ siehe Kapitel 4.2: Überführung eines Systems ins Equilibriums: **Slow Start**

Problem 2: Der Sender schickt ein neues Paket, bevor ein altes das Netz verlassen hat.

→ siehe Kapitel 4.1: Korrektes **Schätzen der Round Trip Time** und korrekte Einstellung des **Retransmit Timers**

Problem 3: Innerhalb des Systems gibt es Ressourcenknappheiten und dadurch kann das Equilibrium nicht erreicht werden.

→ siehe Kapitel 4.3: Mit Multiplicative Decrease auf Überlast reagieren (**Congestion Avoidance**)

→ siehe Kapitel 4.4: Fast Retransmit

Im nächsten Kapitel werden die Methoden und Algorithmen vorgestellt, die sich diesen Problematiken annehmen.

4 Methoden des Congestion Controls

4.1 Round Trip Time Estimation

Ein grundlegendes Problem ist festzustellen, wann es sich um Congestion handelt und wann nicht. Bekommt der Sender keine Bestätigung für ein versendetes Paket, muss er dann davon ausgehen, dass das Netz überlastet ist? Oder befindet es sich noch auf dem Weg und braucht nur länger? Oder ist das Paket verloren gegangen? Wie lange gibt man einem Paket Zeit das Netz zu durchqueren? Und wie lange wartet man bis man das Paket noch einmal sendet?

Der Sender muss abschätzen, wann er davon ausgehen kann, dass das Paket verloren gegangen ist und noch einmal gesendet werden muss. Sendet er das Paket zu früh, befindet sich das alte Paket noch im Netz und das Equilibrium wird gestört. Wartet er zu lange, leidet die Übertragungsrate.

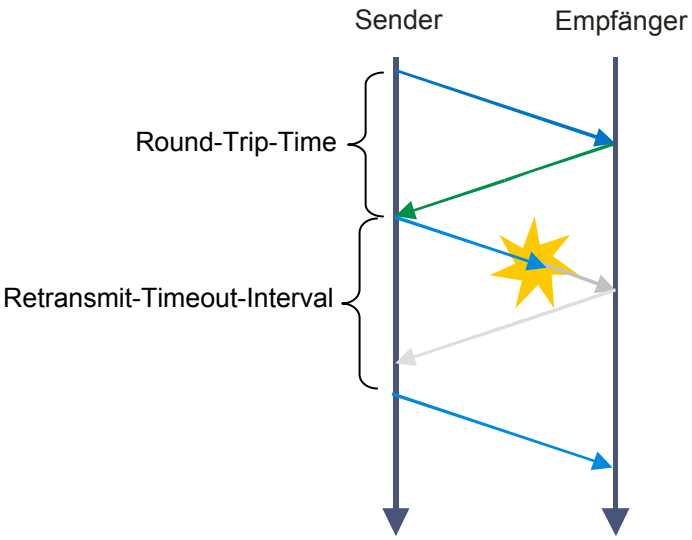

Abbildung 8: Round-Trip-Time und Retransmit-Timeout-Interval

Die Zeit zwischen Versenden und Neuübertragung wird *„Retransmit Timeout Interval“ (RTO)* genannt. Für jedes gesendete Paket wird ein Timer mit dieser Zeit neu gestartet. Läuft er ab (*„timeout“*), wird das Paket neu gesendet (*„retransmission“*). Grundlage zur der Berechnung dieses Timers ist die Rundenzeit (*„Round-Trip-Time“*) eines Pakets. Also die Zeit, die ein Paket braucht, um vom Sender zum Empfänger zu gelangen plus die Zeit, die das Acknowledgement zurück braucht. Doch da man nicht weiß, welche Route ein Paket durch das Netz nimmt, ist es unmöglich vorherzusagen, wie lange ein Paket benötigt. Außerdem variiert die Dauer auch je nach aktueller Belastung des Netzes. Sie muss also geschätzt werden. TCP benutzt hierfür einen adaptiven Algorithmus. Der Timer wird im Verlauf der Verbindung ständig verändert und passt sich der aktuellen Gegebenheiten des Netzes an.

Der Sender verwaltet zur Berechung des Timers zwei Variablen: Die geglättete Rundenzeit (*„Smoothed Round Trip Time“ (SRTT)*) und die Varianz der Rundenzeiten (*„Round-Trip-Time Variation“ (RTTvar)*).

Bevor die erste Rundenzeit gemessen werden kann, wird das Retransmit-Timeout-Interval auf drei Sekunden (empirisch ermittelt) gesetzt.
Nach Messung der ersten Rundenzeit wird die geglättete Rundenzeit auf die gemessene Rundenzeit gesetzt und die Varianz auf die Hälfte der Rundenzeit:

SRTT ← gemessene Rundenzeit

$$\text{RTTvar} \leftarrow \frac{\text{gemessene Rundenzeit}}{2}$$

Das Retransmit-Timeout-Interval wird dann auf

$$\text{RTO} \leftarrow \text{SRTT} + K * \text{RTTvar}$$

gesetzt. K ist hierbei ein Faktor als Sicherheitsspielraum, der auf Erfahrungen beruht und in den meisten Implementierungen auf vier gesetzt.
Fortan wird die Rundenzeit der Pakete kontinuierlich gemessen und nach jeder Messung werden die Werte wie folgt gesetzt:

$$\text{RTTvar} \leftarrow (1 - \beta) * \text{RTTVAR} + \beta * | \text{SRTT} - \text{gemessene Rundenzeit} |$$

$$\text{SRTT} \leftarrow (1 - \alpha) * \text{SRTT} + \alpha * \text{gemessene Rundenzeit}$$

$$\text{RTO} \leftarrow \text{SRTT} + K * \text{RTTvar}$$

Auch hier sind α und β Erfahrungswerte (Von Jacobson [8] vorgeschlagen: $\alpha=1/8$ und $\beta=1/4$). Eine ausschlaggebende Beobachtung bei der Berechung der Varianzen der Rundenzeiten war, dass bei starker Belastung des Netzes die Rundenzeiten stark variieren. Ein Lösungsansatz war, die Standardabweichung der Rundenzeiten in die Berechnung mit einfließen zu lassen. Allerdings ist die Berechnung der Standartabweichung für jedes Paket zu aufwendig. Daher ist die Berechnung der mittlere Abweichung (| SRTT – gemessene Rundenzeit |) sinvoller.

Die Erfahrung hat gezeigt, dass sich so berechnet ein realistischer Wert für den Timer ermitteln lässt, der auch resistent gegen Veränderungen der Netzauslastung ist. Gleichzeitig ist der Timer ein sehr guter Indikator für Congestion, da man mit hoher Wahrscheinlichkeit sagen kann, dass es bei einem „timeout“ um ein Anzeichen für Überlastung des Netzwerkes handelt.

4.2 Slow Start

Dieselben Faktoren, die ein System im Eqilibrium stabilisieren, machen es schwierig, das Equilibrium zu erreichen: Damit Pakete versendet werden können, benötigt das System Acknowledgments vom Empfänger, die es takten. Um Acknowledgments zu bekommen, müssen Daten versendet wurden sein.

Von Jacobson wir in [8] ein Algorithmus vorgestellt, der das System schrittweise ins Equlibrium überführt. Ziel ist, dass dies möglichst schnell passiert, um die verfügbare Bandbreite effektiv zu nutzen. Obwohl der vorgestellte Algorithmus „*Slow-Start*“ genannt wird, erhöht er die Datenmenge im Fluss bis hin zum maximalen Belastbarkeit sehr schnell.

Der Algorithmus bedient sich einer einfachen Idee: Die Menge der Daten im Netz ist abhängig von der Größe des Sendefensters des Senders. Vergrößert man das Fenster, werden mehr Datenpakete aufs Netzwerk gegeben. Verkleinert man es, werden weniger Daten auf das Netzwerk gegeben.

Aus diesem Grund wird auf Seiten des Senders eine weitere Variable eingeführt das „*Congestion Window*“*(cwnd)*. Zu Beginn eines Datentransfers wird das Congestion Window auf die maximale Segmentgröße des Senders („*Sender maximum Segment Size (SMSS)*“)gesetzt[3]. Die maximale Segmentgröße des Senders gibt die maximale Menge an Nutzdaten innerhalb eines TCP-Segments an[4].

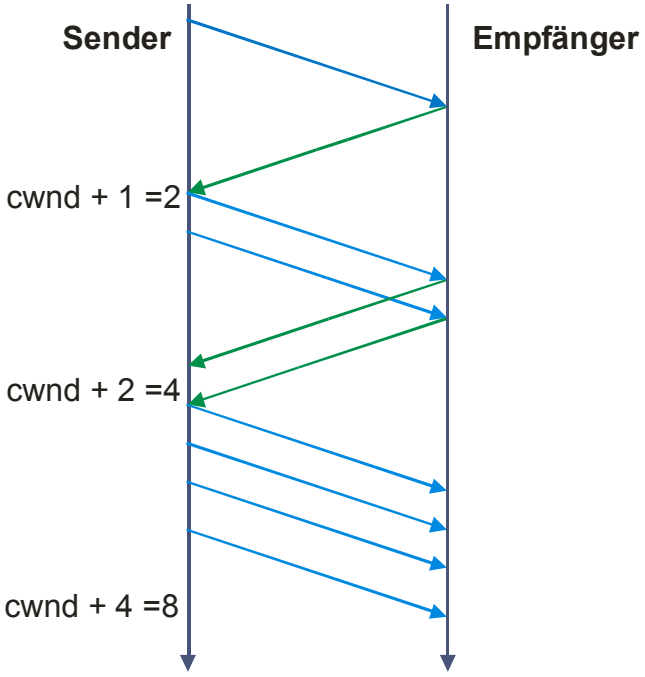

Abbildung 9: Slow Start

Das Sendefenster entspricht zu Beginn in seiner Grösse dem Congestion Window und sendet ein maximales Segment. Im Folgenden wird für jedes empfangene Acknowledgement das Congestion Window um ein Segment erhöht. Und das Sendefenster wird auf das Minimum des Congestion Windows und des „*Advertised Windows“ (awnd*) gesetzt. Das „Advertised Window“ ist ein Wert, der den freien Platz im Empfangspuffer angibt.[5] Es wird so lange gesendet bis entweder ein Anzeichen für Überlastung (Paketverluste (Retransmission-Time-Out)) auftritt oder ein Schwellwert (*„Threshold (ssthresh)“*) überschritten wird. Der Schwellwert kann zu Beginn beliebig hoch gesetzt werden. Oft wird der Wert des Advertised Windows benutzt.

[3] Der Startwert für cwnd kann bis zu 2 SMSS gesetzt werden (nach RFC 2581).
[4] Die SMSS wird durch die Implementierung des TCP/IP-Stacks bestimmt und damit durch das Betriebssystem vorgegeben (1024 Byte bei den meisten Implementierungen). Unter anderem richtet sie sich auch nach der „Maximum Transmission Unit“ (MTU).
[5] Der aktuelle Wert des awnd wird mit jedem ACK übermittelt.

Der Algorithmus:

```
cwnd = 1*SMSS
FOR EACH received ACK DO
    cwnd = cwnd + SMSS
    wnd = min (cwnd , rwnd)
UNTIL (cwnd <  ssthresh) OR (Congestion)
```

Das Congestion Window nimmt expotentiell zu und so wird schnell der Maximalwert ermittelt, bis zu dem gesendet werden kann, ohne dass Überlastungserscheinungen auftreten.

Der Algorithmus kann auf zwei mögliche Arten terminieren. Entweder der Schwellwert wird erreicht oder es tritt Überlastung ein. Je nachdem welcher Fall eintritt, wird unterschiedlich fortgefahren. Registriert der Sender den Verlust von Datenpaketen durch den Retransmission-Timer, wird der Threshold auf die Hälfte der noch ausstehenden Segmente des Sendefenstern verringert (maximal die Hälfte des Congestion Windows) und Slowstart wird neu gestartet. Terminiert der Algorithmus, weil der Threshold erreicht wurde, geht die Verbindung in eine neue Phase über und der „*Congestion Avoidance*“-Algorithmus wird gestartet.

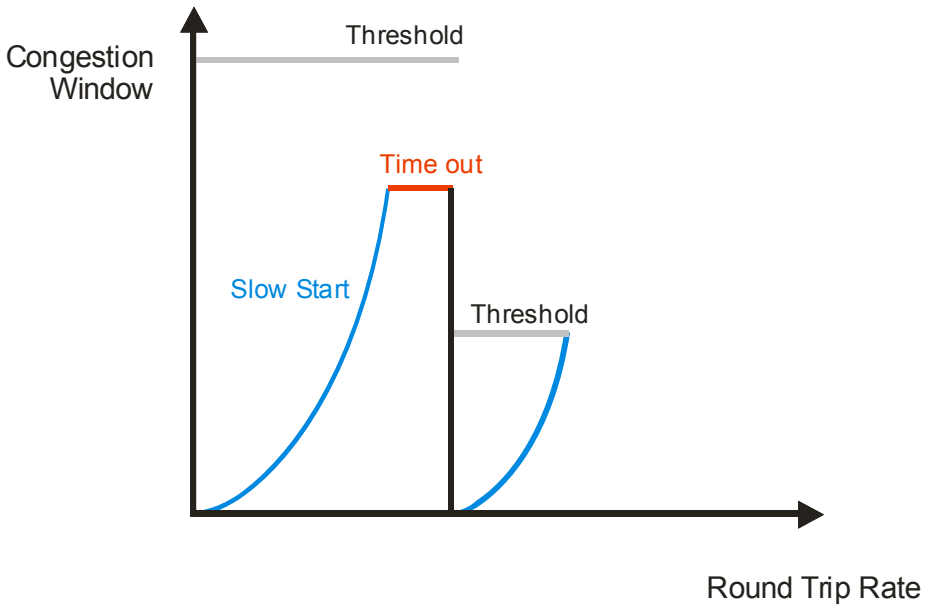

Abbildung 10: Verlauf von Slow Start

4.3 Congestion Avoidance

Congestion Avoidance ist mit Slow Start eng verbunden. Trotzdem ist es ein unabhängiger Algorithmus mit komplett anderen Zielen. Durch Slow Start wurde die obere Grenze der Belastbarkeit einer Verbindung abgesteckt. Aber dieses Optimum kann sich verschieben. Nun versucht man sich langsam an diese Grenze heranzutasten. Congestion Avoidance realisiert dieses langsame Herantasten, indem es das Congestion Window im Gegensatz zu Slow Start linear erhöht.

Pro Round-Trip-Time wird das Fenster um ein Segment erhöht („*additive increase*“). Das entspricht:

$$\text{Für jedes empfangene ACK:} \quad \text{cwnd} = \text{cwnd} + \text{SMSS} * \frac{\text{SMSS}}{\text{cwnd}}$$

Sobald es zu ersten Überlastungsanzeichen kommt terminiert der Algorithmus und das Congestion Window wird halbiert, man sagt auch es wird multiplikative verringert („*multiplicative decrease*“). Man bezeichnet dieses Vorgehen als „*Additive Increase and Multiplicative Decrease*“ *(AIMD)*. Neben den Congestion Window wird auch der Threshold verringert. Er wird auf die Hälfte des Congestion Windows gesetzt.

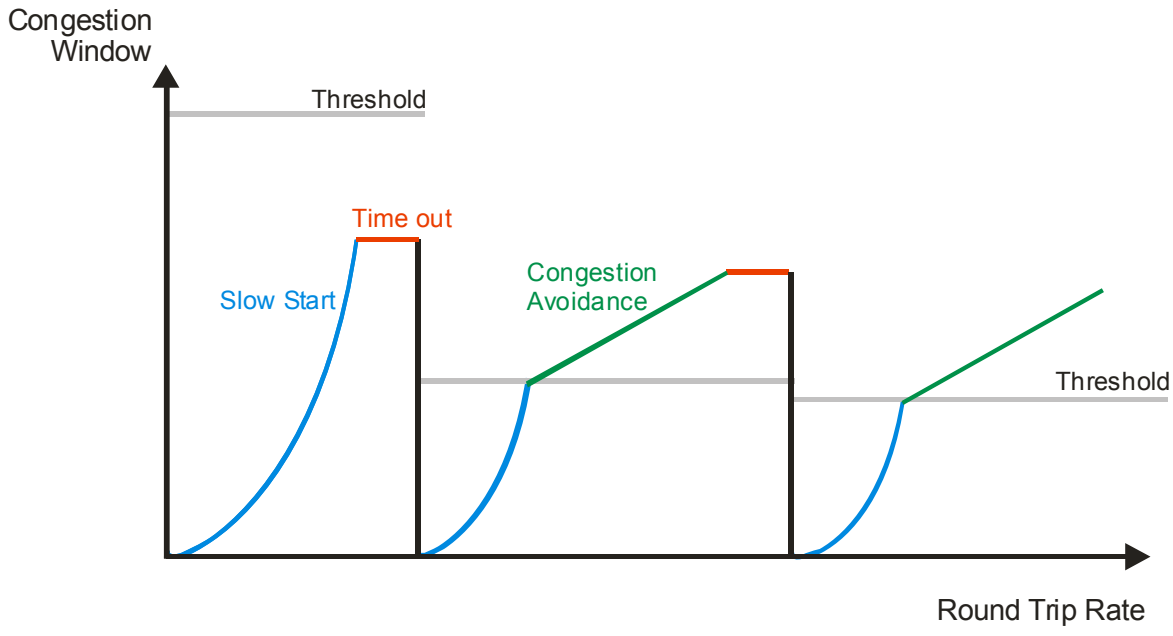

Abbildung 11: Verlauf Slow Start und Congestion Avoidance

4.4 Fast Retransmit & Fast Recovery

Ausser dem Retransmission-Timout gibt es weitere Anzeichen für Überlastung. Ein weiteres Signal ist der Erhalt dreier „*Duplicate-ACKs*“. Wie bereits erwähnt, wird im Acknowledgements des Empfängers mit Hilfe der Sequenznummer codiert, welches Paket er als nächstes erwartet. Ein Duplicate-ACK ist ein Hinweis, dass er ein Datenpaket „out-of-order“ erhalten hat. Das heißt, er hat ein Paket erhalten, dessen Sequenznummer nicht der erwarteten Nummer entspricht. Daraufhin schickt der Empfänger die Bestätigung des letzten erhaltenen Pakets noch einmal und teilt dadurch dem Sender die „Lücke“ und die Sequenznummer des fehlenden Pakets mit. Drei solcher Hinweise sind ein sicheres Signal, dass das Paket verloren gegangen sein muss. Allerdings zeigt dies auch, dass offensichtlich noch Datenpakete übermittelt werden, sonst würde der Empfänger keine ACKs generieren.

Ohne Fast Retransmit werden nach Ablauf des Retransmission-Timers das verlorengegangene Datenpaket und alle nachfolgenden Datenpakete im Sendefenster neu übertragen. Dies bedeutet aber eine unnötige Belastung der Netzwerkressourcen, da alle nachfolgenden Datenpakete auch noch einmal übertragen werden müssen und unnötiger Zeitverlust, da bis zum timeout mit der Neuübertragung gewartet wird, obwohl schon nach den dritten Duplicate-ACK klar ist, dass das Paket verlorengegangen ist.

Durch Fast Retransmit wird das verlorengegangene Paket erneut versendet ohne auf das Ablaufen des Timers zu warten.

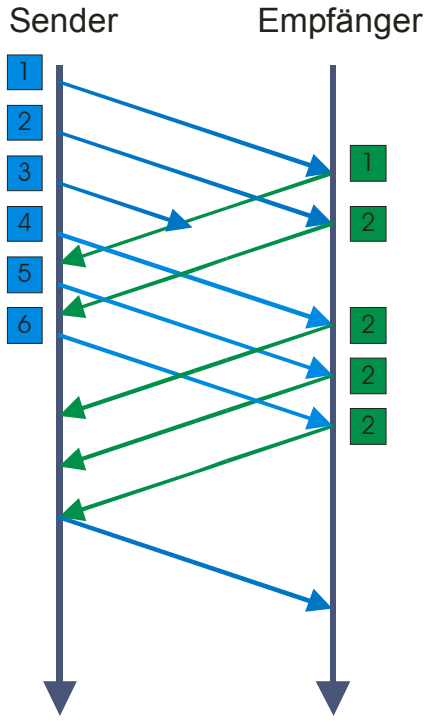

Abbildung 12: Fast Retransmit

Obwohl der Fast Retransmit Algorithmus das verlorengegangene Paket noch einmal gesendet hätte, würde nach Ablauf des Timers ein Slow Start ausgeführt werden bei dem auch die Bestätigten Pakete innerhalb des Fensters noch einmal gesendet werden würden. Daher steuert nach der Fast Retransmission der „*Fast Recovery*“ Algorithmus die Übertragung.

Fast Retransmit und Fast Recovery werden meist wie folgt implementiert:

- Sobald das dritte Duplicate ACK empfangen wurde, wird der Threshold auf die Hälfte der noch ausstehenden Segmente des Sendefensters gesetzt (maximal die Hälfte des Congestion Windows).

  $$\text{ssthresh} = \frac{\text{Menge, der noch ausstehenden Pakete}}{2}$$

- Das verlorengegangene Paket wird erneut übermittelt und das Congestion Window wird neu gesetzt:

  $$\text{cwnd} = \text{ssthresh} + 3\ \text{SMSS}$$

  Das entspricht den drei Segmenten, die durch Duplicate ACKs bestätigt wurden und vom Empfänger gepuffert werden.

- Für jedes weiter empfangene Duplicate ACK wird das Congestion Window um ein SMSS erhöht. Dies wird auch „*inflating*“ genannt.

  $$\text{cwnd} = \text{cwnd} + 1\text{SMSS}$$

  Da der Sender durch das Duplicate ACK weiß, dass das gesendete Paket das Netzwerk verlassen hat, kann er ein weiteres Paket senden. Da aber das verlorengegangene Paket noch nicht bestätigt wurde, „gleiten“ die nachfolgenden Pakete nicht aus dem Fenster. Daher wird es jedes Mal um ein Segment erhöht.

- Dann kann ein neues Segment übermittelt werden, sofern es das Congestion Window und das Empfängerfenster zulassen.

- Sobald der Sender ein „neues“ ACK erhält, also kein Duplicate ACK, wird das Congestion Window wieder auf die Größe des Schwellwertes gesetzt („*deflating*“).

  $$\text{cwnd} = \text{ssthresh}$$

  Durch dieses erste, „neue“ ACK sollten das verlorengegangene Paket und die anschließend empfangenen Pakete „korrekt“ bestätigt werden.

5 Zusammenfassung und Ausblick

TCP sichert den zuverlässigen Datenaustausch im Netzwerk, insbesondere im Internet. Zum zuverlässigen Transfer gehört auch, sich den Gegebenheiten der Verbindung anzupassen und auf mögliche Überlastung zu reagieren. Die Mechanismen der Überlastkontrolle und die

Mechanismen zur Vermeidung der Überlastung, die heute in fast jeder TCP-Ausführung implementiert sind, reagieren auf Signale der Überlastung und verringern automatisch die Belastung des Netzes, indem sie die Datenmenge im Transfer verringern. Gleichzeitig wird versucht, durch kontinuierliches Austesten der Ressourcen den Austausch effizient zu halten.

Nimmt man alle in dieser Arbeit vorgestellten Mechanismen zusammen, ergibt sich das für TCP so typische „Sägezahnmuster“:

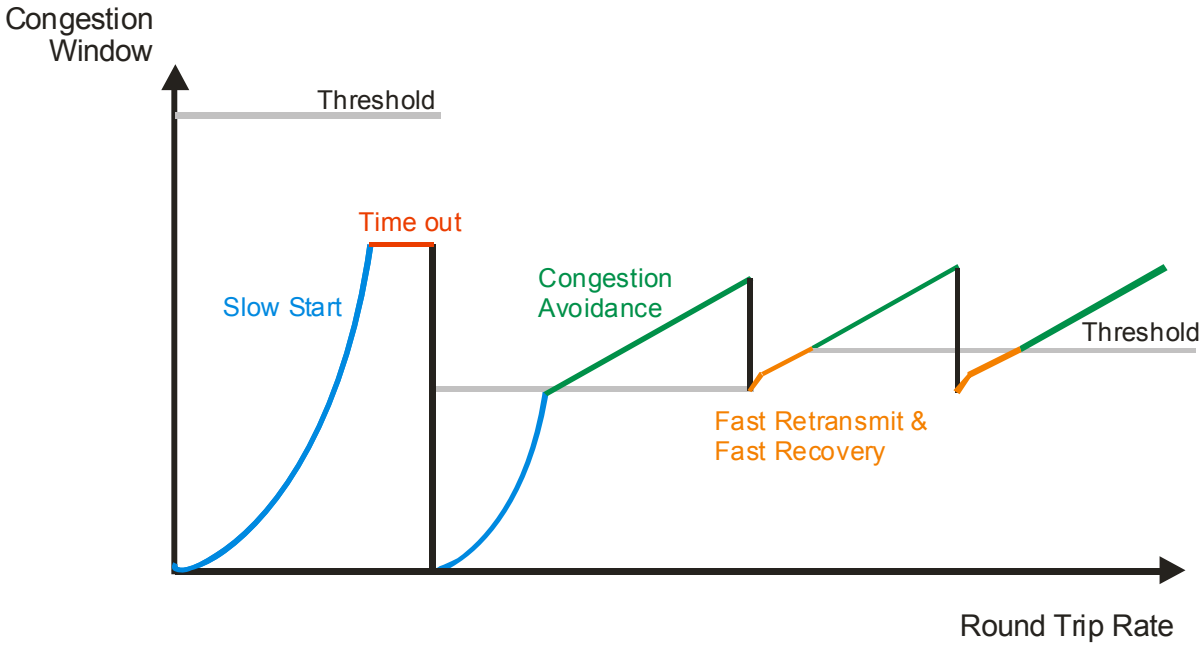

Abbildung 13: TCP Sägezahnverlauf

Die Liste der vorgestellten Mechanismen zur Überlastkontrolle ist sicherlich nicht vollständig. Es gibt viele weitere Ansätze die versuchen, die Zuverlässigkeit einer TCP-Verbindung zu verbessern und die Überlasterscheinungen im Netzwerk zu minimieren. Und das nicht ohne Grund: Eine Untersuchung von Servern unter der Belastung vieler Verbindungen (siehe [3]) hat gezeigt, dass die Recovery-Mechanismen zur Behandlung verlorengegangener Pakete, die in TCP implementiert wurden, noch unzureichend sind. Fast 50% aller verlorenen Segmente müssen nach einem Timeout durch Slow Start neu gesendet werden. Weniger als 45% werden durch Fast Retransmit abgefangen. Es wurde des Weiteren beobachtet, dass es bei Verbindungen unter Belastung zu Verdichtungen der ACKs kommen kann, die das System dadurch falsch „takten“. Auch wurde festgestellt, dass Clients, die mehrere parallele Verbindungen zum Server benutzen, deutlich aggressiver sind als Clients mit nur einer TCP-Verbindung. D.h., dem Client wird mehr Bandbreite zugesprochen als ihm zusteht.
Für viele dieser Probleme gibt es schon Lösungsansätze, die aber noch nicht in den TCP-Standard integriert wurden. Viele andere Problematiken des „Congestion Avoidance and Controls“ sind noch lange nicht gelöst und warten darauf erkannt und gelöst zu werden.

6 Abbildungsverzeichnis

7 Quellenverzeichnis

[1] Allman, M., Paxson , V., *Computing TCP's Retransmission Timer* , RFC 2988, November 2000.

[2] Allman, M., Paxson, V. and R. Stevens, *TCP Congestion Control*, RFC 2581, April 1999.

[3] Balakrishnan, H, Padmanabhan V. N., Srinivasan, S., Stemm, M., Katz, R. H., *TCP Behavior of a Busy Internet Server: Analysis and Improvements*, IEEE Infocom, March 1998.

[4] Comer D.E., Stevens D.L., *Internetworking with TCP/IP, Vol. I - Priciples, Protocols and Internals*. Prentice Hall, Englewood Cliffs, New Jersey, 1995.

[5] Comer D.E., Stevens D.L., *Internetworking with TCP/IP, Vol. II - Design, Implementation and Internals*. Prentice Hall, Englewood Cliffs, New Jersey, 1994.

[6] Floyd, S., *Congestion Control Principles*, RFC 2914, September 2000.

[7] Floyd, S. and T. Henderson, *The NewReno Modification to TCP's Fast Recovery Algorithm*, RFC 2582, April 1999.

[8] Jacobson, V., Karels, M. J., *Congestion Avoidance and Control*", In Proceedings of ACM SIGCOMM '88, Stanford, CA, August 1988.

[9] Leischner, M., Skript zur Veranstaltung *Internet- und E-Businesskommunikation,* http://www.inf.fh-bonn-rhein-sieg.de/person/professoren/leischner/02s/02s-komm/pdf, 23.07.2003

[10] Löb, H.-P., Wilms, D.,*TCP – Transmission Control Protocol*, Grundlagen der Rechnerkommunikation, http://goethe.ira.uka.de/seminare/grk/tcp/, 23.07.2003

[11] Postel, J., *Transmission Control Protocol*, STD 7, RFC 793, September 1981.

[12] Washburn, K., Evans, J., *TCP/IP- Aufbau und Betrieb eines TCP/IP-Netzes*, Addison-Wesley, 1994