Transmission Control Protocol

aus Wikipedia, der freien Enzyklopädie

Das **Transmission Control Protocol** (**TCP**) (zu deutsch *Übertragungssteuerungsprotokoll*) ist eine Vereinbarung (Protokoll) darüber, auf welche Art und Weise Daten zwischen Computern ausgetauscht werden sollen. Nahezu sämtliche aktuellen Betriebssysteme moderner Computer beherrschen TCP und nutzen es für den Datenaustausch mit anderen Rechnern. Das Protokoll ist ein zuverlässiges, verbindungsorientiertes, paketvermittelndes Transportprotokoll in Computernetzwerken. Es ist Teil der Internetprotokollfamilie, der Grundlage des Internets.

Entwickelt wurde TCP von Robert E. Kahn und Vinton G. Cerf. Ihre Forschungsarbeit, die sie im Jahre 1973 begannen, dauerte mehrere Jahre. Die erste Standardisierung von TCP erfolgte deshalb erst im Jahre 1981 als *RFC 793*. Danach gab es viele Erweiterungen, die bis heute in neuen RFCs, einer Reihe von technischen und organisatorischen Dokumenten zum Internet, spezifiziert werden.

Im Unterschied zum verbindungslosen UDP (*User Datagram Protocol*) stellt TCP eine Verbindung zwischen zwei Endpunkten einer Netzverbindung (Sockets) her. Auf dieser Verbindung können in beide Richtungen Daten übertragen werden. TCP setzt in den meisten Fällen auf das IP (Internet-Protokoll) auf, weshalb häufig (und oft nicht ganz korrekt) auch vom „TCP/IP-Protokoll“ die Rede ist. Es ist in Schicht 4 des OSI-Referenzmodells angesiedelt.

TCP (Transmission Control Protocol)	
Familie:	Internetprotokollfamilie
Einsatzgebiet:	Zuverlässiger bidirektionaler Datentransport

TCP im TCP/IP-Protokollstapel:

Anwendung	HTTP	SMTP	…		
Transport	**TCP**				
Internet	IP (IPv4, IPv6)				
Netzzugang	Ethernet	Token Bus	Token Ring	FDDI	…

Standards:	RFC 793 (1981) RFC 1323 (1992)

Aufgrund seiner vielen angenehmen Eigenschaften (Datenverluste werden erkannt und automatisch behoben, Datenübertragung ist in beiden Richtungen möglich, Netzüberlastung wird verhindert usw.) ist TCP ein sehr weit verbreitetes Protokoll zur Datenübertragung. Beispielsweise wird TCP als fast ausschließliches Transportmedium für das WWW, E-Mail und viele andere populäre Netzdienste verwendet.

Inhaltsverzeichnis

Allgemeines

TCP ist im Prinzip eine Ende-zu-Ende-Verbindung in Vollduplex, welche die Übertragung der Informationen in beide Richtungen zur selben Zeit zulässt, analog zu einem Telefongespräch. Diese Verbindung kann auch als zwei Halbduplexverbindungen, bei denen Informationen in beide Richtungen (allerdings nicht gleichzeitig) fließen können, betrachtet werden. Die Daten in Gegenrichtung können dabei zusätzliche Steuerungsinformationen enthalten. Die Verwaltung dieser Verbindung sowie die Datenübertragung werden von der TCP-Software übernommen. Die TCP-Software ist üblicherweise im Netz-Protokollstack des Betriebssystems angesiedelt. Anwendungsprogramme benutzen eine Schnittstelle dazu, meist Sockets, die sich (je nach Betriebssystem unterschiedlich) beispielsweise bei Microsoft Windows in extra einzubindenden Programmbibliotheken („Winsock.dll“ bzw. „wsock32.dll“) oder bei Linux BSD-Sockets. Anwendungen, die diese Software häufig nutzen, sind zum Beispiel Webbrowser und Webserver.

Jede TCP-Verbindung wird eindeutig durch zwei Endpunkte identifiziert. Ein Endpunkt stellt ein geordnetes Paar dar, bestehend aus IP-Adresse und Port. Ein solches Paar bildet eine bidirektionale Software-Schnittstelle und wird auch als Socket bezeichnet. Mit Hilfe der IP-Adressen werden die an der Verbindung beteiligten Rechner identifiziert; mit Hilfe der Ports werden dann auf den beiden beteiligten Rechnern die beiden miteinander kommunizierenden Prozesse identifiziert.

Ein Server wartet beispielsweise auf Port 80 auf eingehende Verbindungen (*listen*) und setzt bei einem Verbindungsaufbau durch einen Client (*accept*) die Verbindung auf einem anderen Socket fort. Dadurch ist es möglich, dass ein Webserver gleichzeitig mehrere Verbindungen zu verschiedenen Rechnern geöffnet hat. Mehrfaches *listen* auf demselben Port ist nicht möglich. Auf der Client-Seite wird eine beliebige Portnummer ab 1024 zugewiesen.

Ports sind 16-Bit-Zahlen (Portnummern) und reichen von 0 bis 65535. Ports von 0 bis 1023 sind reserviert[1] und werden von der IANA vergeben, z. B. ist Port 80 für das im WWW verwendete HTTP reserviert. Das Benutzen der vordefinierten Ports ist nicht bindend. So kann jeder Administrator beispielsweise einen FTP-Server (normalerweise Port 21) auch auf einem beliebigen anderen Port laufen lassen.

Verbindungsaufbau und -abbau

Ein Server, der seinen Dienst anbietet, erzeugt einen Endpunkt (Socket) mit der Portnummer und seiner IP-Adresse. Dies wird als *passive open*[2] oder auch als *listen*[3] bezeichnet.

Will ein Client eine Verbindung aufbauen, erzeugt er einen eigenen Socket aus seiner Rechneradresse und einer eigenen, noch freien Portnummer. Mit Hilfe eines ihm bekannten Ports und der Adresse des Servers kann dann eine Verbindung aufgebaut werden. Eine TCP-Verbindung definiert sich stets eindeutig aus den vier Angaben:

- Quell-IP-Adresse
- Quell-Port
- Ziel-IP-Adresse
- Ziel-Port

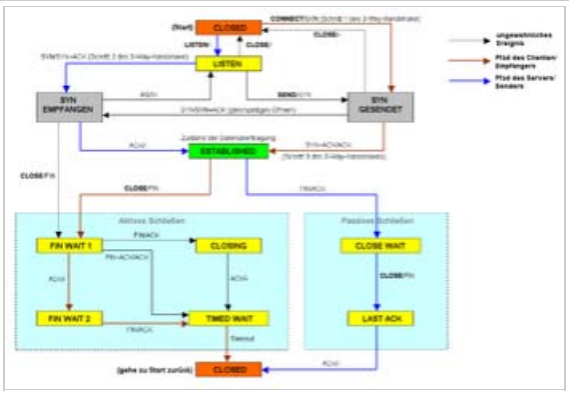


Abb. 2: Verwaltung der TCP-Verbindungen als endlicher Automat

Während der Datenübertragungsphase (*active open*) sind die Rollen von Client und Server (aus TCP-Sicht) vollkommen symmetrisch. Insbesondere kann jeder der beiden beteiligten Rechner einen Verbindungsabbau einleiten.

Während des Abbaus kann die Gegenseite noch Daten übertragen, die Verbindung kann also *halb*-offen sein.

Verbindungsaufbau

Der Client, der eine Verbindung aufbauen will, sendet dem Server ein *SYN*-Paket (von englisch *synchronize*) mit einer Sequenznummer *x*. Die Sequenznummern sind dabei für die Sicherstellung einer vollständigen Übertragung in der richtigen Reihenfolge und ohne Duplikate wichtig. Es handelt sich also um ein Paket, dessen *SYN-Bit* im Paketkopf gesetzt ist (siehe TCP-Header). Die Start-Sequenznummer ist eine beliebige Zahl, deren Generierung von der jeweiligen TCP-Implementierung abhängig ist. Sie sollte jedoch möglichst zufällig sein, um Sicherheitsrisiken zu vermeiden.[4]

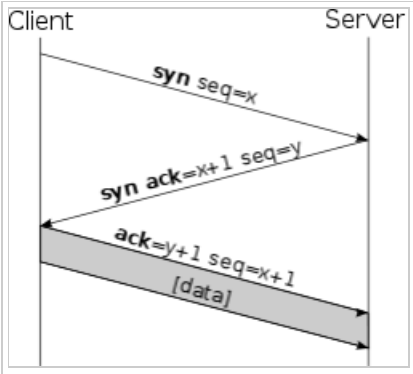


Abb. 3: TCP-Handshake

Der Server (siehe Skizze) empfängt das Paket. Ist der Port geschlossen, antwortet er mit einem TCP-RST, um zu signalisieren, dass keine Verbindung aufgebaut werden kann. Ist der Port geöffnet, bestätigt er den Erhalt des ersten SYN-Pakets und stimmt dem Verbindungsaufbau zu, indem er ein SYN/ACK-Paket zurückschickt (ACK von engl. *acknowledgement* ‚Bestätigung‘). Das gesetzte ACK-Flag im TCP-Header kennzeichnet diese Pakete, welche die Sequenznummer *x+1* des SYN-Pakets im Header enthalten. Zusätzlich sendet er im Gegenzug seine Start-Sequenznummer *y*, die ebenfalls beliebig und unabhängig von der Start-Sequenznummer des Clients ist.

Der Client bestätigt zuletzt den Erhalt des SYN/ACK-Pakets durch das Senden eines eigenen ACK-Pakets mit der Sequenznummer *x+1*. Dieser Vorgang wird auch als „Forward Acknowledgement“ bezeichnet. Aus Sicherheitsgründen sendet der Client den Wert *y+1* (die Sequenznummer des Servers + 1) im ACK-Segment zurück. Die Verbindung ist damit aufgebaut.

Einmal aufgebaut, ist die Verbindung für beide Kommunikationspartner gleichberechtigt, man kann einer bestehenden Verbindung auf TCP-Ebene nicht ansehen, wer der Server und wer der Client ist. Daher hat eine Unterscheidung dieser beiden Rollen in der weiteren Betrachtung keine Bedeutung mehr.

1.	SYN-SENT	→	<SEQ=100><CTL=SYN>	→	SYN-RECEIVED
2.	SYN/ACK-RECEIVED	←	<SEQ=300><ACK=101><CTL=SYN,ACK>	←	SYN/ACK-SENT
3.	ACK-SENT	→	<SEQ=101><ACK=301><CTL=ACK>	→	ESTABLISHED

Verbindungsabbau

Der geregelte Verbindungsabbau erfolgt ähnlich. Statt des SYN-Bits kommt das FIN-Bit (von engl. *finish* ‚Ende‘, ‚Abschluss‘) zum Einsatz, welches anzeigt, dass keine Daten mehr vom Sender kommen werden. Der Erhalt des Pakets wird wiederum mittels ACK bestätigt. Der Empfänger des FIN-Pakets sendet zuletzt seinerseits ein FIN-Paket, das ihm ebenfalls bestätigt wird.

Zudem ist ein verkürztes Verfahren möglich, bei dem FIN und ACK genau wie beim Verbindungsaufbau im selben Paket untergebracht werden. Die *maximum segment lifetime* (MSL) ist die maximale Zeit, die ein Segment im Netzwerk verbringen kann, bevor es verworfen wird. Nach dem Senden des letzten ACKs wechselt der Client in einen zwei MSL andauernden Wartezustand (Waitstate), in dem alle verspäteten Segmente verworfen werden. Dadurch wird sichergestellt, dass keine verspäteten Segmente als Teil einer neuen Verbindung fehlinterpretiert werden. Außerdem wird eine korrekte Verbindungsterminierung sichergestellt. Geht ACK *y+1* verloren, läuft beim Server der Timer ab, und das LAST_ACK-Segment wird erneut übertragen.

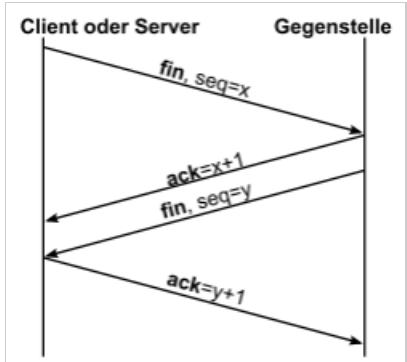


Abb. 4: TCP-Teardown

Der Drei-Wege-Handschlag

Sowohl beim Verbindungsaufbau als auch beim Verbindungsabbau werden die Antworten auf das erste SYN- bzw. FIN-Paket typischerweise zu einem einzelnen Paket (SYN/ACK bzw. FIN/ACK) zusammengefasst – theoretisch wäre auch das Versenden zweier separater Pakete denkbar. Da in diesem Fall nur noch drei Pakete versendet werden müssen, spricht man auch häufig vom sogenannten Drei-Wege-Handschlag. Das Zusammenfassen des FIN-Paketes und des ACK-Paketes ist allerdings problematisch, da das Senden eines FIN-Paketes die Bedeutung hat „es folgen keine weiteren Daten mehr“. Allerdings kann der Sender des FIN-Paketes weiterhin Daten empfangen (wollen); man spricht von einer halboffenen Verbindung (die Empfangsrichtung ist weiterhin offen, während die Senderichtung geschlossen wurde). Es wäre z. B. denkbar, den Beginn einer HTTP-Anfrage (*HTTP-Request*) direkt im SYN-Paket mitzuschicken, weitere Daten, sobald die Verbindung aufgebaut wurde, und im letzten HTTP-Request-Paket die (Senderichtung der) Verbindung gleich mittels FIN zu schließen. In der Praxis wird dieses Verfahren allerdings nicht angewendet. Würde der Browser die Verbindung auf diese Art sofort schließen, würde wahrscheinlich auch der Server die Verbindung schließen anstatt die Anfrage vollständig zu beantworten.

Aufbau des TCP-Headers

Allgemeines

Das TCP-Segment besteht immer aus zwei Teilen, dem Header und der Nutzlast (*Payload*). Die Nutzlast enthält die zu übertragenden Daten, die wiederum Protokollinformationen der Anwendungsschicht wie HTTP oder FTP entsprechen können. Der Header enthält für die Kommunikation erforderliche Daten sowie das Dateiformat beschreibende Information. Den schematischen Aufbau des TCP-Headers kann man im Bild rechts sehen. Da das Options-Feld in der Regel nicht genutzt wird, hat ein typischer Header eine Größe von 20 Byte. Die Werte werden in Byte-Reihenfolge (Big-Endian) angegeben.

Erläuterung

Source Port (Quellport) (2 Byte)
: Gibt die *Portnummer* auf der Senderseite an.

Destination Port (Zielport) (2 Byte)
: Gibt die *Portnummer* auf der Empfängerseite an.

Sequence Number (4 Byte)
: *Sequenznummer* des ersten Daten-Oktetts (*Byte*) dieses TCP-Paketes oder die *Initialisierungs-Sequenznummer* falls das SYN-Flag gesetzt ist. Nach der Datenübertragung dient sie zur Sortierung der TCP-Segmente, da diese in unterschiedlicher Reihenfolge beim Empfänger ankommen können.

Acknowledgement Number (Quittierungsnummer) (4 Byte)
: Sie gibt die *Sequenznummer* an, die der Empfänger dieses TCP-Segmentes als nächstes erwartet. *Sie ist nur gültig, falls das ACK-Flag gesetzt ist.*

Data Offset (4 Bit)
: *Länge des TCP-Headers* in 32-Bit-Blöcken – ohne die Nutzdaten (*Payload*). Hiermit wird die Startadresse der Nutzdaten angezeigt.

Reserved (6 Bit)
: Das *Reserved*-Feld wird nicht verwendet und muss Null sein.

Control-Flags (6 Bit)
: Sind zweiwertige Variablen mit den möglichen Zuständen *gesetzt* und *nicht gesetzt*, die zur Kennzeichnung bestimmter für die Kommunikation und Weiterverarbeitung der Daten wichtiger Zustände benötigt werden. Im Folgenden werden die Flags des TCP-Headers und die von ihrem Zustand abhängigen, auszuführenden Aktionen beschrieben.

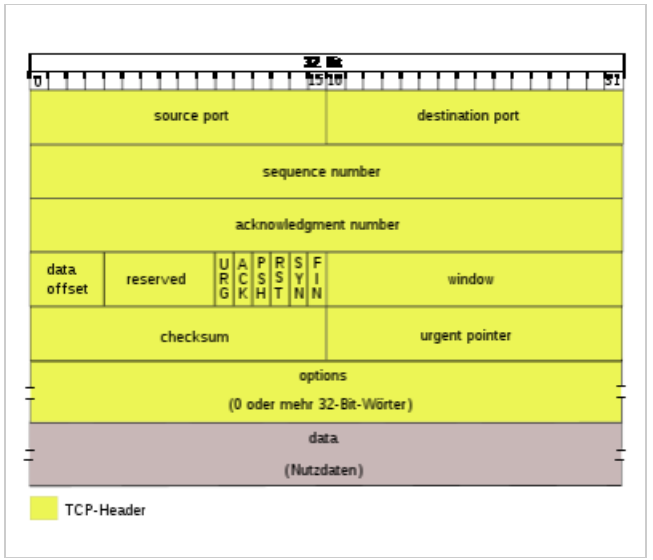


Abb. 5: Aufbau des TCP-Headers

URG
: Ist das Urgent-Flag *(urgent = dringend)* gesetzt, so werden die Daten nach dem Header sofort von der Anwendung bearbeitet. Dabei unterbricht die Anwendung die Verarbeitung der Daten des aktuellen TCP-Segments und liest alle Bytes nach dem Header bis zu dem Byte, auf das das *Urgent-Pointer*-Feld zeigt, aus. Dieses Verfahren ist fern verwandt mit einem Softwareinterrupt. Dieses Flag kann zum Beispiel verwendet werden, um eine Anwendung auf dem Empfänger abzubrechen. Das Verfahren wird nur äußerst selten benutzt, Beispiele sind die bevorzugte Behandlung von CTRL-C (Abbruch) bei einer Terminalverbindung über rlogin oder telnet.

ACK
: Das *Acknowledgment*-Flag hat in Verbindung mit der *Acknowledgment*-Nummer die Aufgabe, den Empfang von TCP-Segmenten beim Datentransfer zu bestätigen. Die *Acknowledgment*-Nummer ist nur gültig, wenn das Flag gesetzt ist.

PSH
: Beim Versenden von Daten über das TCP werden zwei Puffer verwendet. Senderseitig übermittelt die Applikation die zu sendenden Daten an das TCP und dieses puffert die Daten um mehrere kleine Übertragungen effizienter in Form einer einzigen großen zu senden. Nachdem die Daten dann an den Empfänger übermittelt wurden, landen sie im empfängerseitigen Puffer. Dieser verfolgt ähnliche Ziele. Wenn vom TCP mehrere einzelne Pakete empfangen wurden, ist es besser diese zusammengefügt an die Applikation weiterzugeben.
: RFC 1122 und RFC 793 spezifizieren das PSH-Flag so, dass bei gesetztem Flag sowohl der ausgehende, als auch der eingehende Puffer übergangen wird. Da man bei TCP keine Datagramme versendet, sondern einen Datenstrom hat, hilft das PSH-Flag den Strom effizienter zu verarbeiten, da die empfangende Applikation so gezielter aufgeweckt werden kann und nicht bei jedem eintreffenden Datenfragment feststellen muss, dass Teile der Daten noch nicht empfangen wurden, die aber nötig wären um überhaupt weitermachen zu können.
: Hilfreich ist dies, wenn man zum Beispiel bei einer Telnet-Sitzung einen Befehl an den Empfänger senden will. Würde dieser Befehl erst im Puffer zwischengespeichert werden, so würde dieser (stark) verzögert abgearbeitet werden.
: Das PSH-Flag kann, abhängig von der TCP-Implementation im Verhalten zu obiger Erklärung abweichen.

RST
: Das *Reset*-Flag wird verwendet, wenn eine Verbindung abgebrochen werden soll. Dies geschieht zum Beispiel bei technischen Problemen oder zur Abweisung unerwünschter Verbindungen (wie etwa nicht geöffneten Ports, hier wird anders als bei UDP kein ICMP-Paket mit "Port Unreachable" verschickt).

SYN
: Pakete mit gesetztem SYN-Flag initiieren eine Verbindung. Der Server antwortet normalerweise entweder mit SYN+ACK, wenn er bereit ist, die Verbindung anzunehmen, andernfalls mit RST. Dient der Synchronisation von *Sequenznummern* beim Verbindungsaufbau (daher die Bezeichnung SYN).

FIN
: Dieses Schlussflag (*finish*) dient zur Freigabe der Verbindung und zeigt an, dass keine Daten mehr vom Sender kommen. Die FIN- und SYN-Flags haben Sequenznummern, damit diese in der richtigen Reihenfolge abgearbeitet werden.

(Receive) Window (2 Byte)
: Ist die Anzahl der Daten-Oktetts (*Bytes*), beginnend bei dem durch das *Acknowledgmentfeld* indizierten Daten-Oktett, die der Sender dieses TCP-Paketes bereit ist zu empfangen.

Checksum (2 Byte)
: Die *Prüfsumme* dient zur Erkennung von Übertragungsfehlern und wird über den TCP-Header, die Daten und einen Pseudo-Header berechnet. Dieser Header besteht aus der Ziel-IP, der Quell-IP, der TCP-Protokollkennung (0x0006) und der Länge des TCP-Headers inkl. Nutzdaten (in Bytes).

Urgent Pointer (2 Byte)
: Zusammen mit der Sequenz-Nummer gibt dieser Wert die genaue Position des letzten Bytes der Urgent-Daten im Datenstrom an. Die Urgent-Daten beginnen sofort nach dem Header. Der Wert ist nur gültig, wenn das URG-Flag gesetzt ist.

Options (0–40 Byte)
: Das Options-Feld ist unterschiedlich groß und enthält Zusatzinformationen. Die Optionen müssen ein Vielfaches von 32 Bit lang sein. Sind sie das nicht, muss mit Null-Bits aufgefüllt werden (Padding). Dieses Feld ermöglicht, Verbindungsdaten auszuhandeln, die nicht im TCP-Header enthalten sind, wie zum Beispiel die Maximalgröße des Nutzdatenfeldes.

Datenübertragung

TCP-/IP-Paket-Größe

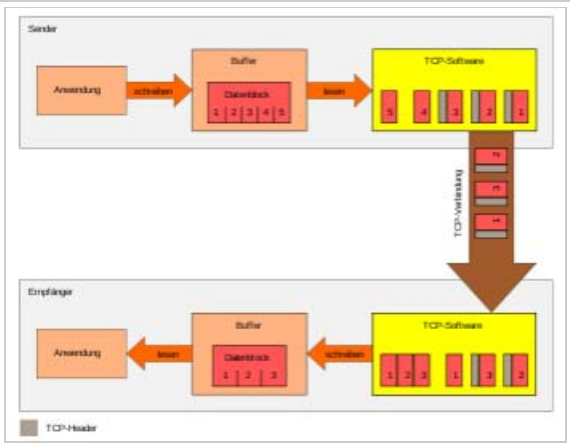


Abb. 6: Segmentierung der Nutzdaten

Ein TCP-Segment hat typischerweise eine Größe von maximal 1500 Bytes. Ein TCP-Segment muss jedoch in die darunter liegende Übertragungsschicht passen, das Internetprotokoll (IP); siehe hierzu auch Maximum Transmission Unit.

IP-Pakete wiederum sind zwar theoretisch bis 65.535 Bytes (64 KiB) spezifiziert, werden aber selbst meist über Ethernet übertragen, und bei Ethernet ist die Rahmengröße (wenn man von *Jumbo Frames* absieht) auf 1500 Bytes festgelegt. TCP- und IP-Protokoll definieren jeweils einen Header von 20 Bytes Größe. Für die Nutzdaten bleiben in einem TCP/IP-Paket also 1460 Bytes (= 1500 Bytes [Nutzdaten] − 20 Bytes − 20 Bytes) übrig. Da die meisten Internet-Anschlüsse DSL verwenden, kommt dort zusätzlich noch das Point-to-Point Protocol (PPP) zwischen IP und Ethernet zur Anwendung, was weitere acht Bytes für den PPP-Rahmen verbraucht. Die Nutzdaten reduzieren sich also auf insgesamt 1500 − 20 − 20 − 8 = 1452 Bytes MSS (Maximum Segment Size). Dies entspricht einer Nutzdatenrate von 96,8 %.

Aufteilen der Anwendungsdaten auf TCP-/IP-Pakete

Empfänger und Sender einigen sich vor dem Datenaustausch über das Options-Feld auf die Größe der MSS. Die Anwendung, die Daten versenden möchte, etwa ein Webserver, legt zum Beispiel einen 10 Kilobyte großen Datenblock im Puffer ab. Um mit einem 1460 Byte großen Nutzdatenfeld 7 Kilobyte Daten zu versenden, teilt die TCP-Software die Daten auf mehrere Pakete auf, fügt einen TCP-Header hinzu und versendet die TCP-Segmente. Dieser Vorgang wird Segmentierung genannt. Der Datenblock im Puffer wird in fünf Segmente aufgeteilt (siehe Abb. 6). Jedes Segment erhält durch die TCP-Software einen TCP-Header. Die TCP-Segmente wurden nacheinander abgeschickt. Diese kommen beim Empfänger nicht notwendigerweise in der gleichen Reihenfolge an wie sie versendet wurden, da im Internet unter Umständen jedes TCP-Segment einen anderen Weg nimmt. Damit die TCP-Software im Empfänger die Segmente wieder sortieren kann, ist jedes Segment nummeriert. Bei der Zuordnung der Segmente im Empfänger wird die Sequenznummer herangezogen.

Die TCP-Software des Empfängers bestätigt diejenigen TCP-Segmente, die einwandfrei (das heißt mit korrekter Prüfsumme) angekommen sind. Andernfalls werden die Pakete neu angefordert.

Beispiel einer TCP-/IP-Datenübertragung

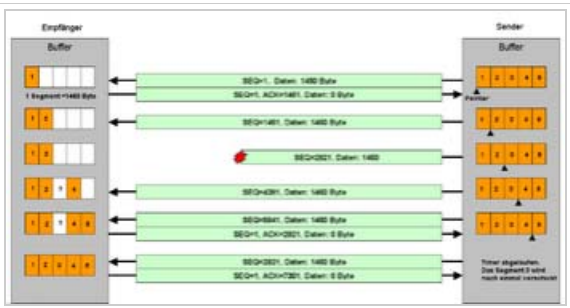


Abb. 7: Beispiel eines Datentransfers

Der Sender schickt sein erstes TCP-Segment mit einer Sequenznummer SEQ=1 (variiert) und einer Nutzdatenlänge von 1460 Byte an den Empfänger. Der Empfänger bestätigt es mit einem TCP-Header ohne Daten mit ACK=1461 und fordert damit das zweite TCP-Segment ab dem Byte Nummer 1461 beim Sender an. Dieser schickt es dann mit einem TCP-Segment und SEQ=1461 an den Empfänger. Dieser bestätigt es wieder mit einem ACK=2921 und so weiter. Der Empfänger braucht nicht jedes TCP-Segment zu bestätigen, wenn diese zusammenhängend sind. Empfängt er die TCP-Segmente 1–5, so braucht er nur das letzte TCP-Segment zu bestätigen. Fehlt zum Beispiel das TCP-Segment 3, weil es verlorengegangen ist, so kann er nur die 1 und die 2 bestätigen, 4 und 5 jedoch noch nicht. Da der Sender keine Bestätigung für die 3 bekommt, läuft sein Timer ab, und er verschickt die 3 noch einmal. Kommt die 3 beim Empfänger an, so bestätigt er alle fünf TCP-Segmente. Der Sender startet für jedes TCP-Segment, welches er auf die Reise schickt, einen Timer (RTT).

Retransmission Timer

Zur Feststellung, wann ein Paket im Netzwerk verloren gegangen ist, wird vom Sender ein Timeout verwendet, bis zu dem das ACK der Gegenseite eingetroffen sein muss. Ein zu niedriger Timeout bewirkt, dass Pakete, die eigentlich korrekt angekommen sind, wiederholt werden; ein zu hoher Timeout bewirkt, dass bei tatsächlichen Verlusten das zu wiederholende Paket unnötig spät gesendet wird. Aufgrund unterschiedlicher Laufzeiten der zugrundeliegenden IP-Pakete ist nur ein dynamisch an die Verbindung angepasster Timer sinnvoll. Die Details werden von RFC 2988 wie folgt festgelegt:

- Der Timeout (RTO = Retransmission Timeout) berechnet sich aus zwei beim Sender mitgeführten Statusvariablen:
  - der geschätzten Round Trip Time (SRTT = Smoothed RTT)
  - sowie deren Varianz (RTTVAR).
- Initial wird geschätzt, dass RTO = 3s.
- Nach der Messung der RTT des ersten gesendeten Pakets wird gesetzt:
  - SRTT := RTT
  - RTTVAR := 0,5 * RTT
  - RTO := RTT + 4 * RTTVAR (Sollte 4 * RTTVAR kleiner sein als die Messgenauigkeit des Timers, wird stattdessen diese addiert.)
- Bei jeder weiteren Messung der RTT' werden die Werte aktualisiert:
  - SRTT := (1-α) * SRTT + α * RTT' (Es wird somit nicht einfach die neue RTT' gesetzt, sondern diese mit einem Faktor α geglättet. Es wird empfohlen, α = 1/8 zu wählen.)

- RTTVAR := (1-β) * RTTVAR + β * |SRTT – RTT'| (Auch die Varianz wird mit einem Faktor β geglättet; da die Varianz eine durchschnittliche Abweichung angibt (welche immer positiv ist), wird hier der Betrag der Abweichung von geschätzter und tatsächlicher RTT' verwendet, nicht die einfache Differenz. Es wird empfohlen, beta = 1/4 zu wählen.)
- RTO := SRTT + 4 * RTTVAR (Sollte 4*RTTVAR kleiner sein als die Messgenauigkeit des Timers, wird stattdessen diese addiert. Für den RTO gilt – unabhängig von der Berechnung – ein Minimalwert von 1 s; es darf auch ein Maximalwert vergeben werden, sofern dieser mindestens 60 s beträgt.)

Durch die Wahl von 2er-Potenzen (4 bzw. 1/2, 1/4 etc.) als Faktoren, können die Berechnungen in der Implementierung durch einfache Shift-Operationen realisiert werden.

Zur Messung der RTT muss Karns Algorithmus verwendet werden; d.h. es werden nur diejenigen Pakete zur Messung verwendet, deren Bestätigung eintrifft, ohne dass das Paket zwischendurch erneut gesendet wurde. Der Grund dafür ist, dass bei einer erneuten Übertragung nicht klar wäre, welches der wiederholt gesendeten Pakete tatsächlich bestätigt wurde, so dass eine Aussage über die RTT eigentlich nicht möglich ist.

Wurde ein Paket nicht innerhalb des Timeouts bestätigt, so wird der RTO verdoppelt (sofern er noch nicht die optionale obere Schranke erreicht hat). In diesem Fall dürfen (ebenfalls optional) die für SRTT und RTTVAR gefundenen Werte auf ihren Anfangswert zurückgesetzt werden, da sie möglicherweise die Neuberechnung der RTO stören könnten.

Zusammenhang von Flusssteuerung und Staukontrolle

In den folgenden zwei Abschnitten werden die TCP-Konzepte zur Flusssteuerung und Staukontrolle (oder Überlaststeuerung) erläutert. Dabei werden das *Sliding Window* und das *Congestion Window* eingeführt. Der Sender wählt als tatsächliche Sendefenstergröße das Minimum aus beiden Fenstern.

Flusssteuerung

Da die Anwendung Daten aus dem Puffer liest, ändert sich der Füllstand des Puffers ständig. Deshalb ist es notwendig, den Datenfluss dem Füllstand entsprechend zu steuern. Dies geschieht mit dem *Sliding Window* und dessen Größe. Den Puffer des Senders erweitern wir, wie in Abb. 8 zu sehen, auf 10 Segmente. In der Abb. 8a werden gerade die Segmente 1–5 übertragen. Die Übertragung ist vergleichbar mit Abb. 7. Obwohl der Puffer des Empfängers in Abb. 7 am Ende voll ist, fordert er mit ACK=7301 die nächsten Daten ab dem Byte 7301 beim Sender an. Dies hat zur Folge, dass das nächste TCP-Segment vom Empfänger nicht mehr verarbeitet werden kann. Ausnahmen sind jedoch TCP-Segmente mit gesetztem URG-Flag. Mit dem Window-Feld kann er dem Sender mitteilen, dass er keine Daten mehr verschicken soll. Dies geschieht, indem er im Window-Feld den Wert Null einträgt (Zero Window). Der Wert Null entspricht dem freien Speicherplatz im Puffer. Die Anwendung des Empfängers liest nun die Segmente 1–5 aus dem Puffer, womit wieder ein Speicherplatz von 7300 Byte frei ist. Damit kann er die restlichen Segmente 6–10 mit einem TCP-Header, der die Werte SEQ=1, ACK=7301 und Window=7300 enthält, beim Sender anfordern. Der Sender weiß nun, dass er maximal fünf TCP-Segmente an den Empfänger schicken kann, und verschiebt das Window um fünf Segmente nach rechts (siehe Abb. 8b). Die Segmente 6–10 werden nun alle zusammen als *Burst* verschickt. Kommen alle TCP-Segmente beim Empfänger an, so quittiert er sie mit SEQ=1 und ACK=14601 und fordert die nächsten Daten an.

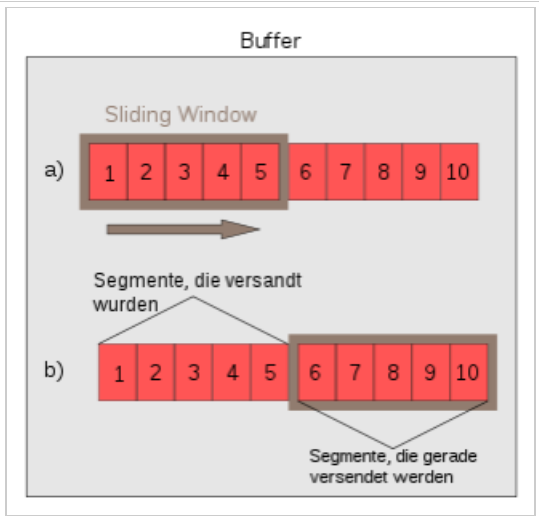


Abb. 8: Sliding Window

Silly Window Syndrome

Der Empfänger sendet ein *Zero Window* an den Sender, da sein Puffer voll ist. Die Anwendung beim Empfänger liest allerdings nur zwei Byte aus dem Puffer. Der Empfänger schickt einen TCP-Header mit Window=2 (Window Update) an den Sender und fordert gleichzeitig die zwei Byte an. Der Sender kommt der Aufforderung nach und schickt die zwei Byte in einem 42 Byte großen Paket (mit IP-Header und TCP-Header) an den Empfänger. Damit ist der Puffer des Empfängers wieder voll, und er schickt wieder ein *Zero Window* an den Sender. Die Anwendung liest jetzt zum Beispiel hundert Byte aus dem Puffer. Der Empfänger schickt wieder einen TCP-Header mit einem kleinen Window-Wert an den Sender. Dieses Spiel setzt sich immer wieder fort und verschwendet Bandbreite, da nur sehr kleine Pakete versandt werden. Clarks Lösung ist, dass der Empfänger ein *Zero Window* sendet und so lange mit dem *Window Update* warten soll, bis die Anwendung mindestens die maximale Segmentgröße (maximum segment size, in unseren bisherigen Beispielen 1460 Byte) aus dem Puffer gelesen hat oder der Puffer halbleer ist – je nachdem, was zuerst eintritt (Dave Clark, 1982). Auch der Sender kann zu kleine Pakete abschicken und dadurch Bandbreite verschwenden. Dieser Umstand wird mit dem Nagle-Algorithmus beseitigt. Deswegen ergänzt er sich mit Clarks Lösung.

Überlaststeuerung/Staukontrolle (Congestion Control)

Im Internet, in dem viele Netze mit unterschiedlichen Eigenschaften verbunden werden, ist Datenverlust einzelner Pakete durchaus normal. Wird eine Verbindung stark belastet, werden immer mehr Pakete verworfen, die entsprechend wiederholt werden müssen. Durch die Wiederholung steigt wiederum die Belastung, ohne geeignete Maßnahmen kommt es zu einem Datenstau.

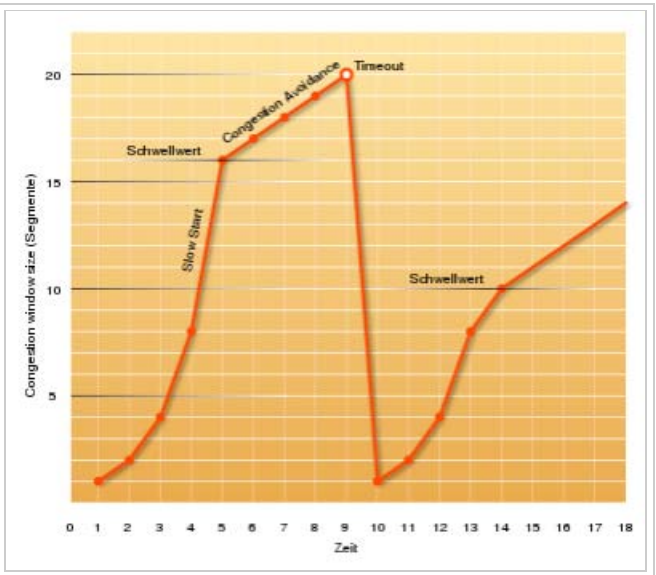
Die Verlustrate wird von einem IP-Netzwerk ständig beobachtet. Abhängig von der Verlustrate wird die Senderate durch geeignete Algorithmen beeinflusst: Normalerweise wird eine TCP/IP-Verbindung langsam gestartet (Slow-Start) und die Senderate schrittweise erhöht, bis es zum Datenverlust kommt. Ein Datenverlust verringert die Senderate, ohne Verlust wird sie wiederum erhöht. Insgesamt nähert sich die Datenrate so zunächst dem jeweiligen zur Verfügung stehenden Maximum und bleibt ungefähr dann dort. Eine Überbelastung wird vermieden.

Algorithmus zur Überlaststeuerung

Gehen bei einer bestimmten Fenstergröße Pakete verloren, kann das festgestellt werden, wenn der Sender innerhalb einer bestimmten Zeit (Timeout) keine Bestätigung (ACK) erhält. Man muss davon ausgehen, dass das Paket aufgrund zu hoher Netzlast von einem Router im Netz verworfen wurde. Das heißt, der Puffer eines Routers ist vollgelaufen; es handelt sich hier sozusagen um einen Stau im Netz. Um den Stau aufzulösen, müssen alle beteiligten Sender ihre Netzlast reduzieren. Dazu werden im RFC 2581 vier Algorithmen definiert: *slow start*, *congestion avoidance*, *fast retransmit* und *fast recovery*, wobei *slow start* und *congestion avoidance* zusammen verwendet werden. Die zwei Algorithmen *fast retransmit* und *fast recovery* werden auch zusammen verwendet und sind eine Erweiterung der Algorithmen *slow start* und *congestion avoidance*.

Slow Start und Congestion Avoidance

Zu Beginn einer Datenübertragung dient der Slow-Start-Algorithmus zur Bestimmung des *congestion window* (wörtlich: Überlastfenster), um einer möglichen Überlastsituation vorzubeugen. Man möchte Staus vermeiden, und da die momentane Auslastung des Netzes nicht bekannt ist, wird mit zunächst kleinen Datenmengen begonnen. Der Algorithmus startet mit einem kleinen Fenster von einer MSS (Maximum Segment Size), in dem Datenpakete vom Sender zum Empfänger übertragen werden.



Grafische Darstellung des Slow-Start-Algorithmus

Der Empfänger sendet nun eine Bestätigung (ACK) an den Sender zurück. Für jedes empfangene ACK wird die Größe des *congestion window* um eine MSS erhöht. Da für jedes versandte Paket bei erfolgreicher Übertragung ein ACK geschickt wird, führt dies innerhalb einer Roundtrip-Zeit zu einer Verdopplung des Congestion Windows. In dieser Phase gibt es also ein exponentielles Wachstum. Wenn das Fenster beispielsweise das Versenden von zwei Paketen gestattet, so erhält der Sender auch zwei ACKs und erhöht das Fenster daher um 2 auf 4. Dieses exponentielle Wachstum wird so lange fortgesetzt, bis der sogenannte *Slow-Start Threshold* erreicht wird (engl. *threshold* ‚Schwelle'). Die Phase des exponentiellen Wachstums wird auch *Slow Start Phase* genannt.

Danach wird das Congestion Window nur noch um eine MSS erhöht, wenn alle Pakete aus dem Fenster erfolgreich übertragen wurden. Es wächst also pro Roundtrip-Zeit nur noch um eine MSS, also nur noch linear. Diese Phase wird als *Congestion Avoidance Phase* bezeichnet. Das Wachstum wird beendet, wenn das vom Empfänger festgelegte Empfangsfenster erreicht worden ist (siehe Fluss-Steuerung).

Kommt es zu einem Timeout, wird das *congestion window* wieder auf 1 zurückgesetzt, und der *Slow-Start Threshold* wird auf die Hälfte des alten Congestion Windows herabgesetzt. Die Phase des exponentiellen Wachstums wird also verkürzt, so dass das Fenster bei häufigen Paketverlusten nur langsam wächst. Das Wachstum des Fensters ist in der Regel exponentiell.

Fast-Retransmit und Fast-Recovery

Fast-Retransmit und *Fast-Recovery* werden eingesetzt, um nach einem Paketverlust schneller auf die Stau-Situation zu reagieren. Dazu informiert ein Empfänger den Sender, wenn Pakete außer der Reihe ankommen und somit dazwischen ein Paketverlust vorliegt. Hierfür bestätigt der Empfänger das letzte korrekte Paket erneut für jedes weitere ankommende Paket außer der Reihe. Man spricht dabei von *Dup-Acks* (*Duplicate Acknowledgments*). Der Sender bemerkt die duplizierten Bestätigungen, und nach dem dritten Duplikat sendet er sofort, vor Ablauf des Timers, das verlorene Paket erneut. Weil nicht auf den Ablauf des Timers gewartet werden muss, heißt das Prinzip *Fast Retransmit*. Die Dup-Acks sind auch Hinweise darauf, dass zwar ein Paketverlust stattfand, aber doch die folgenden Pakete angekommen sind. Deshalb wird das Sendefenster nach dem Fehler nur halbiert und nicht wie beim Timeout wieder mit Slow-Start begonnen. Zusätzlich kann das Sendefenster noch um die Anzahl der Dup-Acks erhöht werden, denn jedes steht für ein weiteres Paket, welches den Empfänger erreicht hat, wenn auch außer der Reihe. Da dadurch nach dem Fehler schneller wieder die volle Sendeleistung erreicht wird, nennt man das Prinzip *Fast-Recovery* („schnelles Erholen").

Bis zum Erkennen des Paketverlustes wurden noch weitere Pakete bis zur Sendefenstergröße übertragen. Der Empfänger konnte diese nicht nutzen, weil ein Paket der Serie verlorenging, aber er kann sie im Puffer halten. Nach der Neuübertragung des verlorenen Pakets durch den Sender bestätigt der Empfänger mittels ACK und einer höheren Sequenznummer die nun vollständige Paketfolge. Das erspart dem Sender, alle nach dem Paketverlust übertragenen Pakete erneut zu übertragen, und er kann sofort mit ganz neuen Paketen fortfahren; man nennt das *kumuliertes ACK*.

Selective ACKs werden genutzt, um noch mehr Kontrollinformationen über den Datenfluss vom Empfänger an den Sender zurückzuschicken. Dabei wird nach einem Paketverlust vom Empfänger im TCP-Optionsfeld ein zusätzlicher Header eingefügt, aus welchem der Sender genau ersehen kann, welche Pakete bereits angekommen sind und welche fehlen (im Gegensatz zu den standardmäßigen kumulativen ACKs von TCP, s. o.). Als bestätigt gelten die Pakete auch weiterhin erst dann, wenn der Empfänger dem Sender ein ACK für die Pakete übermittelt hat.

TCP-Tahoe und TCP-Reno

Bei den nach Orten in Nevada benannten TCP-Congestion-Control-Varianten Tahoe und Reno handelt es sich um zwei verschiedene Verfahren, wie TCP auf ein Überlast-Ereignis in Form von *Timeouts* oder *Dup-Acks* („duplicate ACKs“, also mehrere aufeinanderfolgende Nachrichten, welche dasselbe Datensegment ACKen) reagiert.

Das inzwischen nicht mehr verwendete TCP Tahoe reduziert, sobald ein Timeout vorliegt, das Congestion Window für die nächste Übertragungseinheit auf 1. Anschließend startet wieder der TCP-Slow-Start-Prozess (mit verringertem Threshold, s. u.), bis ein neues Timeout- oder DUP-Acks-Ereignis stattfindet oder aber der Schwellwert (*Threshold*) zum Übergang in die Congestion-Avoidance-Phase erreicht wird. Dieser Schwellwert wurde nach dem Auftreten des Überlast-Ereignisses auf die Hälfte der Größe des derzeitigen Congestion Window gesetzt. Der Nachteil dieses Verfahrens ist zum einen, dass ein Paketverlust nur durch einen Timeout festgestellt wird, mitunter also recht lange dauert, und zum anderen die starke Reduktion des Congestion Windows auf 1.

Die Weiterentwicklung von Tahoe ist TCP-Reno. Hierbei wird zwischen auftretenden Timeout- und Dup-Acks-Ereignissen unterschieden: Während TCP-Reno beim Auftreten eines Timeout genauso verfährt wie TCP Tahoe, wendet es beim Auftreten von drei doppelten Acks eine andere Variante für die Festlegung des nachfolgenden Congestion Windows an. Die grundlegende Idee dabei ist, dass der Verlust eines Segments auf dem Weg zum Empfänger nicht nur durch einen Timeout erkannt werden kann, sondern auch dadurch, dass der Empfänger mehrfach dieselben ACKs für das unmittelbar *vor* dem verlorengegangenen Segment zurückschickt (und zwar jedes Mal, wenn er ein weiteres Segment nach der „Lücke“ empfängt). Daher wird das nachfolgende Congestion Window auf die Hälfte des Wertes des Congestion Windows zum Zeitpunkt des Überlast-Ereignisses gesetzt; anschließend wird wieder in die Congestion Avoidance Phase übergegangen. Dieses Verhalten wird, wie oben im Artikel erwähnt, als *Fast-Recovery* beschrieben.

Überlaststeuerung als Forschungsfeld

Die genaue Gestaltung der TCP-Überlaststeuerung war und ist ein überaus aktives Forschungsfeld mit zahlreichen wissenschaftlichen Publikationen. Auch heute arbeiten weltweit viele Wissenschaftler an Verbesserungen der TCP-Überlaststeuerung oder versuchen, sie an bestimmte äußere Gegebenheiten anzupassen. In diesem Zusammenhang sind insbesondere die speziellen Bedingungen der diversen drahtlosen Übertragungstechniken zu erwähnen, welche oft zu hohen oder stark schwankenden Laufzeitverzögerungen oder zu hohen Paketverlusten führen. TCP geht standardmäßig bei Paketverlusten davon aus, dass der Übertragungsweg an irgendeiner Stelle ausgelastet ist (Datenstau). Dies ist bei drahtgebundenen Netzen auch meistens der Fall, da dort nur selten Pakete *auf der Leitung* verlorengehen, sondern nicht angekommene Pakete fast immer von einem überlasteten Router verworfen wurden. Die richtige Reaktion auf so einen „Datenstau“ ist daher die Reduktion der Senderate. Bei drahtlosen Netzen trifft diese Annahme jedoch nicht mehr zu. Aufgrund des wesentlich unzuverlässigeren Übertragungsmediums treten Paketverluste oft auf, ohne dass einer der Router überlastet ist. In diesem Szenario ist das Reduzieren der Senderate jedoch nicht sinnvoll. Im Gegenteil, eine Erhöhung der Senderate, etwa durch Mehrfachsenden von Paketen, könnte die Zuverlässigkeit der Verbindung erhöhen.

Häufig basieren diese Änderungen bzw. Erweiterungen der Überlastkontrolle auf komplexen mathematischen bzw. regelungstechnischen Fundamenten. Der Entwurf entsprechender Verbesserungen ist alles andere als einfach, da im allgemeinen gefordert wird, dass TCP-Verbindungen mit älteren Überlastkontrollmechanismen durch die neuen Verfahren nicht wesentlich benachteiligt werden dürfen, wenn beispielsweise mehrere TCP-Verbindungen um Bandbreite auf einem gemeinsam genutzten Medium „kämpfen“. Aus all diesen Gründen ist die in der Realität verwendete TCP-Überlaststeuerung auch wesentlich komplizierter gestaltet, als es weiter oben im Artikel beschrieben wird.

Aufgrund der zahlreichen Forschungen zur TCP-Überlaststeuerung setzten sich im Laufe der Zeit verschiedene Überlaststeuerungsmechanismen als Quasi-Standards durch. Hier sind insbesondere TCP Reno, TCP Tahoe und TCP Vegas zu nennen.

Im Folgenden sollen exemplarisch einige neuere bzw. experimentellere Ansätze grob umrissen werden. Ein Ansatz ist beispielsweise RCF (Router Congestion Feedback). Hierbei werden durch die Router entlang dem Pfad umfangreichere Informationen an die TCP-Sender oder -Empfänger geschickt, damit diese ihre Ende-zu-Ende-Überlaststeuerung besser abstimmen können. Hierdurch sind erwiesenermaßen beträchtliche Durchsatzsteigerungen möglich. Beispiele dafür finden sich in der Literatur unter den Stichworten XCP (Explicit Control Protocol), EWA (Explicit Window Adaptation), FEWA (Fuzzy EWA), FXCP (Fuzzy XCP) und ETCP (Enhanced TCP) (Stand: Mitte 2004). Weiterhin ist die Explicit Congestion Notification (ECN) eine Implementierung einer RCF. Vereinfacht gesagt bilden diese Verfahren eine Überlaststeuerung nach Art von ATM nach.

Andere Ansätze verfolgen die logische Trennung der Regelschleife einer TCP-Verbindung in zwei oder mehr Regelschleifen an den entscheidenden Stellen im Netz (z. B. beim sogenannten Split-TCP). Weiterhin gibt es das Verfahren der logischen Bündelung mehrerer TCP-Verbindungen in einem TCP-Sender, damit diese Verbindungen ihre Informationen über den momentanen Zustand des Netzes austauschen und schneller reagieren können. Hier ist insbesondere das Verfahren EFCM (Ensemble Flow Congestion Management) zu nennen. All diese Verfahren können unter dem Begriff *Network Information Sharing* zusammengefasst werden (siehe Weblinks).

TCP-Prüfsumme und TCP-Pseudo-Header

Der Pseudo-Header ist eine Zusammenstellung von Header-Anteilen eines TCP-Pakets und Teilen des darin eingekapselten IP-Headers. Es ist ein Modell, an dem sich die Berechnung der TCP-Prüfsumme (engl. *checksum*) anschaulich beschreiben lässt.

Falls IP mit TCP eingesetzt wird, ist es wünschenswert, den Header des IP-Paketes mit in die Sicherung von TCP aufzunehmen. Dadurch ist die Zuverlässigkeit seiner Übertragung garantiert. Darum bildet man den TCP-Pseudoheader. Er besteht aus: IP-Absender und -Empfängeradresse, einem Null-Byte, einem Byte mit dem Wert 6 und der Länge des TCP-Segments mit TCP-Header. Der Pseudoheader wird für die Berechnung der Prüfsumme vor den TCP-Header gelegt. Anschließend berechnet man die Prüfsumme. Die Summe wird im Feld „checksum“ abgelegt und das Fragment versendet. Kein Pseudoheader wird je versendet.

TCP-Pseudoheader (IPv4)

Bit offset	Bits 0–3	4–7	8–15	16–31
0	IP-Absenderadresse			
32	IP-Empfängeradresse			
64	00000000		6 (=TCP)	TCP-Länge
96	Quellport			Zielport
128	Sequenznummer			
160	ACK-Nummer			
192	Datenoffset	Reserviert	Flags	Window
224	Prüfsumme			Urgent pointer
256	Options (optional)			
256/288+	Daten			

Die Berechnung der Prüfsumme für IPv4 ist in RFC 793 definiert:

Die Prüfsumme ist das 16-Bit-Einerkomplement der Einerkomplement-Summe aller 16-Bit-Wörter im Header und der Nutzdaten des unterliegenden Protokolls. Wenn ein Segment eine ungerade Anzahl Bytes enthält, wird ein Padding-Byte angehängt. Das Padding wird nicht übertragen. Während der Berechnung der Prüfsumme wird das Prüfsummenfeld selbst mit Nullen gefüllt.

Der Empfänger erstellt ebenfalls den Pseudo-Header und führt anschließend die gleiche Berechnung aus, ohne das Checksum-Feld auf Null zu setzen. Dadurch sollte das Ergebnis FFFF (Hexadezimal) sein. Ist dies nicht der Fall, so wird das TCP-Segment ohne Nachricht verworfen. Dies hat zur Folge, dass der RTT-Timer beim Absender abläuft und das TCP-Segment noch einmal abgeschickt wird.

Der Grund für dieses komplizierte Verfahren liegt darin, dass sich Teile des IP-Headers während des Routings im IP-Netz verändern. Das TTL-Feld wird bei jedem IP-Hop um eins dekrementiert. Würde das TTL-Feld in die Prüfsummenberechnung einfließen, würde IP die Sicherung des Transports durch TCP zunichte machen. Deshalb wird nur ein Teil des IP-Headers in die Prüfsummenberechnung einbezogen. Die Prüfsumme ist zum einen wegen ihrer Länge von nur 16 Bit und wegen der einfachen Berechnungsvorschrift anfällig für nicht erkennbare Fehler. Bessere Verfahren wie CRC-32 wurden zur Zeit der Definition als zu aufwendig angesehen.

Datenintegrität und Zuverlässigkeit

Im Gegensatz zum verbindungslosen UDP implementiert TCP einen bidirektionalen, byte-orientierten, zuverlässigen Datenstrom zwischen zwei Endpunkten. Das darunterliegende Protokoll (IP) ist paketorientiert, wobei Datenpakete verlorengehen können, in verkehrter Reihenfolge ankommen dürfen und sogar doppelt empfangen werden können. TCP wurde entwickelt, um mit der Unsicherheit der darunterliegenden Schichten umzugehen. Es prüft daher die Integrität der Daten mittels der Prüfsumme im Paketkopf und stellt die Reihenfolge durch Sequenznummern sicher. Der Sender wiederholt das Senden von Paketen, falls keine Bestätigung innerhalb einer bestimmten Zeitspanne (Timeout) eintrifft. Die Daten der Pakete werden beim Empfänger in einem Puffer in der richtigen Reihenfolge zu einem Datenstrom zusammengefügt und doppelte Pakete verworfen.

Der Datentransfer kann selbstverständlich jederzeit nach dem „Aufbau einer Verbindung“ gestört, verzögert oder ganz unterbrochen werden. Das Übertragungssystem läuft dann in einen Timeout. Der vorab getätigte „Verbindungsaufbau“ stellt also keinerlei Gewähr für eine nachfolgende, dauerhaft gesicherte Übertragung dar.

Bestätigungen

Die jeweilige Länge des Puffers, bis zu der keine Lücke im Datenstrom existiert, wird bestätigt (*Windowing*). Dadurch ist das Ausnutzen der Netz-Bandbreite auch bei großen Strecken möglich. Bei einer Übersee- oder Satellitenverbindung dauert das Eintreffen des ersten ACK-Signals aus technischen Gründen bisweilen mehrere 100 Millisekunden, in dieser Zeit können unter Umständen mehrere hundert Pakete gesendet werden. Der Sender kann den Empfängerpuffer füllen, bevor die erste Bestätigung eintrifft. Alle Pakete im Puffer können gemeinsam bestätigt werden. Bestätigungen können zusätzlich zu den Daten in den TCP-Header des entgegengesetzten Datenstroms eingefügt werden (*Piggybacking*), falls der Empfänger ebenfalls Daten für den Sender bereithält.

Siehe auch

- CYCLADES – Das Vorbild für TCP mit seinem Datagramm CIGALE.
- Liste der standardisierten Ports - enthält offizielle und inoffizielle festgelegte TCP-Ports

Literatur

- Douglas Comer: *Internetworking with TCP/IP. Principles, Protocols, and Architectures.* Prentice Hall, 2000, ISBN 0-13-018380-6.
- Craig Hunt: *TCP/IP Netzwerk-Administration.* O’Reilly, Bejing 2003, ISBN 3-89721-179-3.

- Richard Stevens: *TCP/IP Illustrated.* Volume 1. The Protocols. Addison-Wesley, Boston 1994, 2004. ISBN 0-201-63346-9.
- Richard Stevens: *TCP/IP Illustrated.* Volume 2. The Implementation. Addison-Wesley, Boston 1994, ISBN 0-201-63354-X.
- Andrew S. Tanenbaum: *Computernetzwerke.* Pearson Studium, München 2003, S. 580ff (4. Auf.), ISBN 978-3-8273-7046-4.
- James F. Kurose, Keith W. Ross: *Computernetze. Ein Top-Down-Ansatz mit Schwerpunkt Internet.* Bafög-Ausgabe. Pearson Studium, München 2004, ISBN 3-8273-7150-3.
- Michael Tischer, Bruno Jennrich: *Internet Intern. Technik & Programmierung.* Data-Becker, Düsseldorf 1997, ISBN 3-8158-1160-0.

Weblinks

RFCs

- RFC 793 (Transmission Control Protocol)
- RFC 1071 (Berechnen der Prüfsumme für IP, UDP und TCP)
- RFC 1122 (Fehlerbehebungen bei TCP)
- RFC 1323 (Erweiterungen bei TCP)
- RFC 2018 (TCP SACK – Selective Acknowledgment Options)
- RFC 2581 (TCP Congestion Control – TCP-Überlastkontrolle)
- RFC 3168 (Explicit Congestion Notification)

Sonstige

- Congestion Avoidance and Control (http://www.cs.berkeley.edu/~brewer/cs262/cong-avoid.pdf) (TCP-Meilenstein 1988)
- Warriors of the net (http://www.warriorsofthe.net/) (Film zu TCP)
- Einführung in TCP/IP (http://www.rvs.uni-bielefeld.de/~heiko/tcpip/index.html) – Online-Einführung in die TCP/IP-Protokolle. Heiko Holtkamp, AG Rechnernetze und Verteilte Systeme, Technische Fakultät, Universität Bielefeld.

Einzelnachweise

1. *Port numbers*. (http://www.iana.org/assignments/port-numbers) Internet Assigned Numbers Authority (IANA), 18. Juni 2010, abgerufen am 20. Juni 2010 (englisch).
2. *RFC 793, Seite 11*. (http://tools.ietf.org/html/rfc793#page-11) Internet Engineering Task Force (IETF), September 1981, S. 85, abgerufen am 20. Juni 2010 (englisch): „A passive OPEN request means that the process wants to accept incoming connection requests rather than attempting to initiate a connection."
3. *RFC 793, Seite 21*. (http://tools.ietf.org/html/rfc793#page-21) Internet Engineering Task Force (IETF), September 1981, S. 85, abgerufen am 20. Juni 2010 (englisch): „Briefly the meanings of the states are: LISTEN – represents waiting for a connection request from any remote TCP and port."
4. Steven M. Bellovin: *RFC 1948 – Defending Against Sequence Number Attacks*. (http://tools.ietf.org/html/rfc1948) In: *Internet Engineering Task Force (IETF)*. AT&T Research, Mai 1996, S. 5, abgerufen am 20. Juni 2010 (englisch).

Von „http://de.wikipedia.org/wiki/Transmission_Control_Protocol"
Kategorien: Netzwerkprotokoll (Transportschicht) | TCP/IP