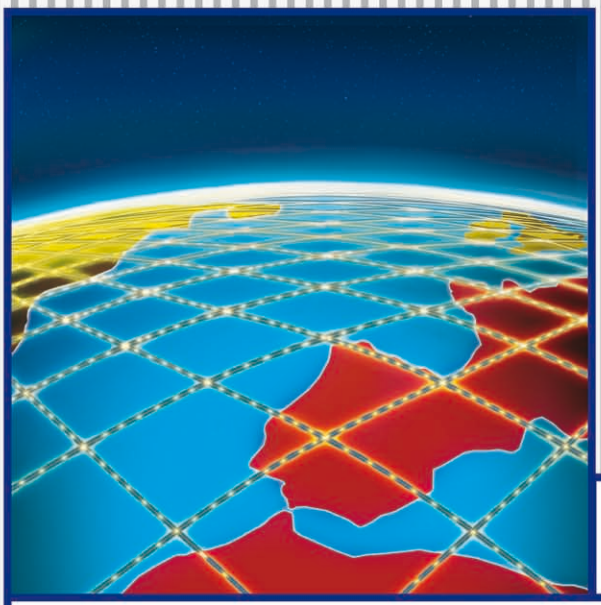
WWW.PEARSON-STUDIUM.DE
CWS
COMPANION WEBSITE



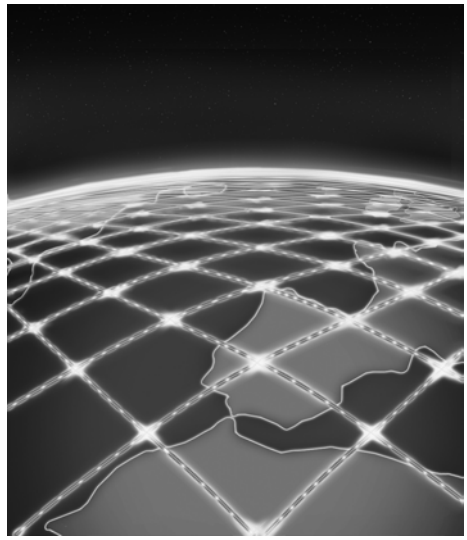
James F. Kurose
Keith W. Ross

Computernetzwerke

Der Top-Down-Ansatz

5., aktualisierte Auflage

ALWAYS LEARNING
PEARSON



James F. Kurose
Keith W. Ross

Computernetzwerke

Der Top-Down-Ansatz

5., aktualisierte Auflage

Fachliche Betreuung
Martin Mauve
Björn Scheuermann

PEARSON

Higher Education
München • Harlow • Amsterdam • Madrid • Boston
San Francisco • Don Mills • Mexico City • Sydney
a part of Pearson plc worldwide

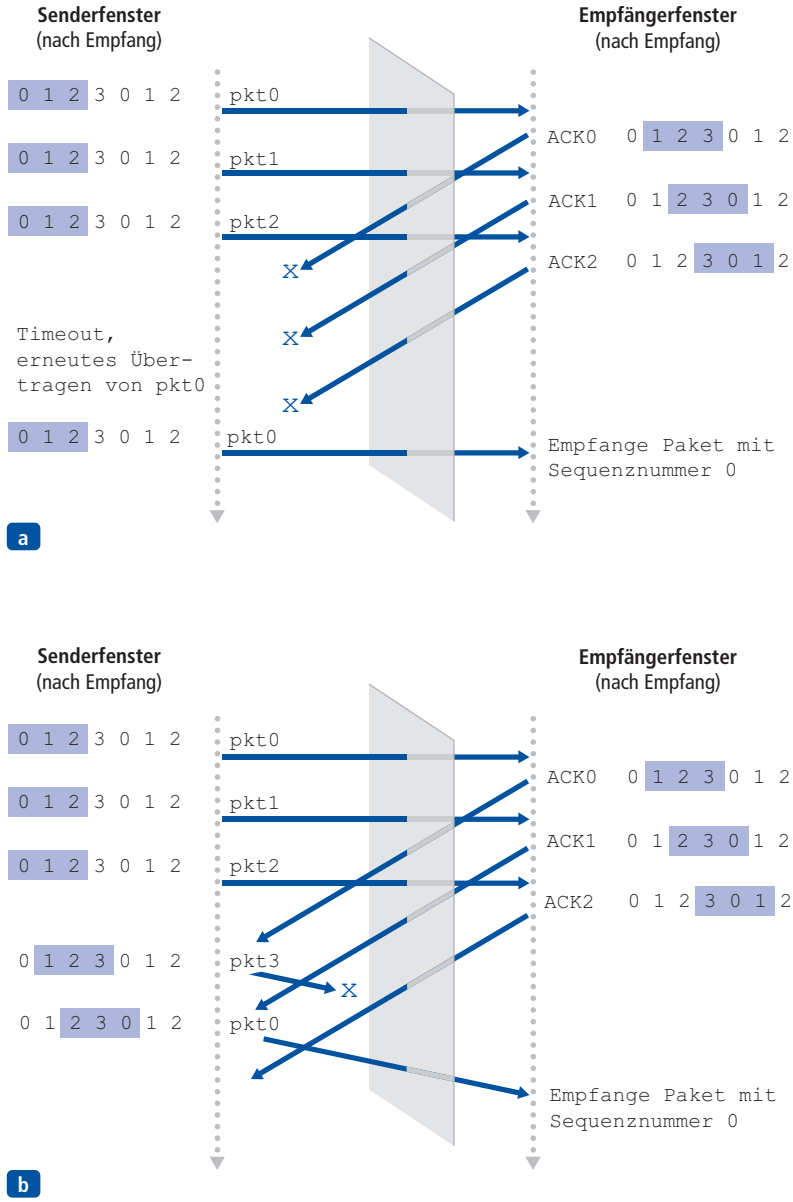


Abbildung 3.27: Dilemma des SR -Empfängers bei zu großen Fenstern: neues Paket oder Übertragungswiederholung?

nicht. Aber wie groß muss das Fenster sein? In einer Übungsaufgabe am Ende des Kapitels sollen Sie zeigen, dass bei SR-Protokollen die Fenstergröße kleiner oder gleich der Hälfte des Sequenznummernbereiches sein muss.

Auf der Webseite zum Buch finden Sie ein Applet, das die Operation des SR-Protokolls anschaulich macht. Versuchen Sie, dieselben Versuche durchzuführen, die Sie

mit dem GBN-Applet gemacht haben. Stimmen die Ergebnisse mit dem überein, was Sie erwarten?

Dies beendet unsere Diskussion zuverlässiger Datenübertragungsprotokolle. Wir haben viele Themenbereiche betrachtet und zahlreiche Mechanismen eingeführt, die zusammen einen zuverlässigen Datentransfer ermöglichen. ▶Tabelle 3.1 fasst diese Mechanismen zusammen. Nun, da wir sie alle in Aktion gesehen haben und in der Lage sind, das Gesamtbild zu erkennen, möchten wir Sie dazu anregen, diesen Abschnitt erneut zu lesen. Sie sollen erkennen, wie diese Techniken schrittweise zusammengefügt wurden, um immer komplexere (und realistischere) Modelle des Kanals zu unterstützen, der Absender und Empfänger verbindet, sowie um die Leistung der Protokolle zu steigern.

Mechanismus	Einsatzzweck, Kommentare
Prüfsumme	Wird verwendet, um Bitfehler in einem gesendeten Paket zu erkennen.
Timer	Wird verwendet, um ein Paket nochmals zu übertragen, möglicherweise weil das Paket (oder das zugehörige ACK) auf dem Kanal verloren ging. Weil Timeouts auftreten können, wenn ein Paket verzögert wird, aber nicht verloren geht (frühzeitiger Timeout), oder weil ein Paket beim Empfänger eingetroffen sein kann, aber das ACK verloren ging, können doppelte Kopien eines Paketes beim Empfänger ankommen.
Sequenznummer	Wird für die fortlaufende Nummerierung von Datenpaketen verwendet, die vom Sender zum Empfänger laufen. Lücken in den Sequenznummern der erhaltenen Pakete erlauben es dem Empfänger, ein verlorenes Paket zu erkennen. Pakete mit doppelten Sequenznummern ermöglichen es dem Empfänger, doppelte Kopien eines Paketes zu erkennen.
Acknowledgment (ACK)	Wird vom Empfänger verwendet, um dem Absender mitzuteilen, dass ein Paket oder ein Satz von Paketen richtig empfangen worden ist. Acknowledgments tragen normalerweise die Sequenznummer des Paketes oder der Pakete, die bestätigt werden. Acknowledgments können je nach Protokoll einzeln oder kumulativ sein.
Negatives Acknowledgment (NAK)	Wird vom Empfänger verwendet, um dem Absender mitzuteilen, dass ein Paket nicht richtig empfangen wurde. Ein negatives Acknowledgment enthält normalerweise die Sequenznummer des Paketes, das nicht richtig empfangen wurde.
Pipelining, Sendefenster	Der Sender darf mehrere unbestätigte Pakete senden, deren Sequenznummern innerhalb eines gegebenen Bereiches, dem Sendefenster, liegen müssen. Indem mehrere Pakete gesendet werden dürfen, aber noch nicht bestätigt werden müssen, kann die Auslastung im Vergleich zum Stop-and-Wait-Ansatz gesteigert werden. Wir werden später sehen, dass die Fenstergröße von verschiedenen Parametern beeinflusst wird, etwa der Fähigkeit des Empfängers, Nachrichten zu empfangen und zu puffern oder der (Über-)Lastsituation im Netz.

Tabelle 3.1: Zusammenfassung von Mechanismen für die zuverlässige Datenübertragung und ihrer Verwendung

Beenden wir unsere Diskussion der zuverlässigen Datentransferprotokolle, indem wir eine verbleibende Annahme in unser zugrunde liegendes Kanalmodell einfügen. Erinnern Sie sich daran, dass wir davon ausgegangen sind, dass Pakete nicht innerhalb des Kanals zwischen Sender und Empfänger umsortiert werden können. Dies ist im Allgemeinen eine vernünftige Voraussetzung, besonders dann, wenn Sender und Empfänger durch eine einzelne physikalische Leitung verbunden sind. Ist jedoch der „Kanal“, der die beiden verbindet, ein komplexes Netzwerk, dann können Pakete umgeordnet werden. Eine Auswirkung umgeordneter Pakete besteht darin, dass alte Kopien eines Paketes mit einer Sequenznummer oder einer Acknowledgment-Nummer von x auftauchen können, obwohl weder das Sender- noch das Empfängerfenster x enthält. Durch das Umordnen der Pakete erscheint der Kanal, als ob er im Wesentlichen Pakete puffert und spontan zu einem beliebigen Zeitpunkt in der Zukunft wieder ausspuckt. Weil Sequenznummern wiederverwendet werden können, muss besondere Sorgfalt darauf verwandt werden, sich vor solchen doppelten Paketen zu schützen. Bei dem in der Praxis durchgeführten Ansatz wird sichergestellt, dass eine Sequenznummer nicht wieder verwendet wird, bis der Absender „sicher“ ist, dass zuvor gesandte Pakete mit der Sequenznummer x nicht länger im Netz sind. Dies erfolgt durch die Annahme, dass ein Paket höchstens eine gewisse feste Zeitspanne im Netzwerk überleben kann. Bei TCP für Hochgeschwindigkeitsnetze wird eine maximale Paketlebensdauer von etwa drei Minuten angenommen [RFC 1323]. [Sunshine 1978] beschreibt eine Methode für die Verwendung von Sequenznummern, welche das Problem des Umordnens völlig vermeidet.

3.5 Verbindungsorientierter Transport: TCP

Nun, da wir die zugrunde liegenden Prinzipien des zuverlässigen Datentransfers behandelt haben, wenden wir uns TCP zu – das verbindungsorientierte, zuverlässige Transportprotokoll des Internets. In diesem Abschnitt werden wir sehen, dass TCP, um zuverlässigen Datentransfer anbieten zu können, auf vielen der Grundlagen aufbaut, die wir im vorherigen Abschnitt erörtert haben, etwa Fehlererkennung, Übertragungswiederholungen, kumulative Bestätigungen, Timer und Header-Felder für Sequenznummern und Acknowledgment-Nummern. TCP wird in RFC 793, RFC 1122, RFC 1323, RFC 2018 und RFC 2581 definiert.

3.5.1 Die TCP-Verbindung

TCP heißt **verbindungsorientiert**, weil zwei Prozesse zunächst einen „Handshake“ durchführen müssen, bevor ein Anwendungsprozess beginnen kann, Daten zum anderen zu senden – d.h., sie müssen zu Beginn eine Reihe von Segmenten austauschen, um die Parameter des folgenden Datentransfers auszuhandeln. Als Teil des TCP-Verbindungsaufbaus initialisieren beide Seiten der Verbindung mehrere TCP-Zustandsvariablen dieser Verbindung (von denen viele in diesem Abschnitt und in Abschnitt 3.7 erörtert werden).

Die TCP-„Verbindung“ ist keine durchgehende TDM- oder FDM-Leitung wie in einem leitungsvermittelten Netzwerk. Es handelt sich auch nicht um eine virtuelle Leitung (siehe Kapitel 1), weil der Zustand der Verbindung ausschließlich in den beiden Endsystemen gehalten wird. Da das TCP-Protokoll nur auf den Endsystemen läuft und nicht in den dazwischen liegenden Netzwerkelementen (Router und Switches der Sicherungsschicht), halten die Netzwerkelemente keinen TCP-Verbindungsstatus. In der Tat sind sich die zwischengeschalteten Router der TCP-Verbindungen gar nicht bewusst; sie sehen Datagramme, nicht Verbindungen.

Eine TCP-Verbindung bietet einen **Vollduplexdienst**: Besteht eine TCP-Verbindung zwischen Prozess A auf einem Host und Prozess B auf einem anderen Host, dann können Anwendungsschichtdaten sowohl von Prozess A zu Prozess B als auch von Prozess B zu Prozess A fließen. Eine TCP-Verbindung ist außerdem immer eine Punkt-zu-Punkt-Verbindung, besteht also zwischen einem einzelnen Sender und einem einzelnen Empfänger. Sogenanntes „Multicasting“ (siehe Abschnitt 4) – der Datentransfer von einem Sender zu vielen Empfängern in einer einzelnen Sendeoperation – ist mit TCP nicht möglich. Bei TCP sind drei Hosts einer zu viel!

Werfen wir nun einen Blick darauf, wie eine TCP-Verbindung aufgebaut wird. Nehmen Sie an, dass ein Prozess, der auf einem Host läuft, eine Verbindung zu einem anderen Prozess auf einem anderen Host initiieren will. Erinnern Sie sich daran, dass der Prozess, der die Verbindung initiiert, als *Client-Prozess* und der andere Prozess als Server-Prozess bezeichnet wird. Der Client-Anwendungsprozess informiert die Client-Trans-

Fallstudie

Vinton Cerf, Robert Kahn und TCP/IP

In den frühen 1970ern wuchs die Zahl der Paketvermittlungsnetze stark an, wobei das ARPAnet – der Vorläufer des Internets – nur eines von vielen Netzen war. Jedes dieser Netze hatte sein eigenes Protokoll. Zwei Forscher, Vinton Cerf und Robert Kahn, erkannten die Wichtigkeit, diese Netze zusammenzuschalten, und entwickelten ein netzwerkübergreifendes Protokoll, das sie TCP/IP (für Transmission Control Protocol/Internet Protocol) nannten. Während Cerf und Kahn ihr Protokoll zunächst als eine Einheit betrachteten, wurden später die Bestandteile TCP und IP getrennt. Cerf und Kahn veröffentlichten im Mai 1974 in den IEEE Transactions on Communications Technology [Cerf 1974] einen Artikel über TCP/IP.

Das TCP/IP-Protokoll, die Basis des heutigen Internets, wurde vor den modernen PCs und Workstations entworfen, bevor Ethernet und lokale Netzwerke Verbreitung gefunden hatten, vor dem Web, Audio-/Video-Streaming und Chat. Cerf und Kahn erkannten die Notwendigkeit eines Netzwerkprotokolls, das einer Vielzahl von noch zu definierenden Anwendungen eine solide Basis bot, und gleichzeitig beliebige Kombinationen von Hosts und Sicherungsschichtprotokollen ermöglichte.

2004 erhielten Cerf und Kahn den ACM Turing Award, den „Nobelpreis der Informatik“, für ihre „Pionierarbeiten im Internetworking, einschließlich des Entwurfs und der Implementation der grundlegenden Kommunikationsprotokolle des Internets, TCP/IP, und für ihre inspirierte Führungsrolle im Bereich Computernetzwerke“.

portschicht zunächst, dass er eine Verbindung zu einem Prozess im Server einleiten will. Erinnern Sie sich an Abschnitt 2.7: Ein Java Client-Programm erreicht dies durch den Befehl

```
Socket clientSocket = new Socket("hostname", portNumber);
```

wobei `hostname` der Name des Servers ist und `portNumber` den Prozess auf dem Server festlegt. Die Transportschicht auf dem Client fährt dann fort, eine TCP-Verbindung mit der TCP-Instanz auf dem Server einzuleiten. Am Ende dieses Abschnittes erörtern wir detailliert das Verfahren, mit dem die Verbindung aufgebaut wird. Jetzt genügt uns zu wissen, dass der Client zuerst ein spezielles TCP-Segment sendet. Der Server antwortet mit einem zweiten speziellen TCP-Segment und der Client antwortet zuletzt wieder mit einem dritten speziellen Segment. Die ersten beiden Segmente tragen keine Nutzlast, d.h. keine Anwendungsschichtdaten. Das dritte dieser Segmente kann Nutzlast tragen, muss es aber nicht. Weil drei Segmente zwischen beiden Hosts ausgetauscht werden, wird dieses Verfahren zum Eröffnen einer Verbindung oft **Drei-Wege-Handshake** *(three-way handshake)* genannt.

Ist einmal eine TCP-Verbindung hergestellt, können beide Anwendungsprozesse einander Daten senden. Lassen Sie uns das Senden von Daten vom Client-Prozess an den Server-Prozess betrachten. Der Client-Prozess überträgt einen Datenstrom durch den Socket (die Tür des Prozesses), wie in Abschnitt 2.7 beschrieben. Sobald die Daten durch diese Tür gehen, sind sie in Händen des auf dem Client laufenden TCP. Wie ▶Abbildung 3.28 zeigt, packt TCP diese Daten in den **Sendepuffer** der Verbindung, einen der Puffer, die während des anfänglichen Drei-Wege-Handshakes reserviert wurden. Von Zeit zu Zeit holt TCP Teile der Daten aus dem Sendepuffer. Interessanterweise ist die TCP-Spezifikation [RFC 793] sehr zurückhaltend hinsichtlich der Zeitpunkte, zu denen TCP gepufferte Daten senden sollte und legt nur fest, dass TCP „diese Daten in Segmenten nach eigenem Gutdünken senden“ sollte. Die Datenmenge, die auf einmal in ein Segment gepackt werden darf, wird durch die **maximale Segmentgröße** (**MSS**, *maximum segment size*) beschränkt. Die *MSS* wird normalerweise gesetzt, indem zuerst die Länge des größtmöglichen Rahmens der Sicherungsschicht, der vom lokalen sendenden Host ausgesandt werden kann, bestimmt wird (die sogenannte **maximale Übertragungseinheit**, **MTU**, *maximum transmission unit*).

Danach wird die *MSS* so gesetzt, dass ein TCP-Segment (nach dem Verkapseln in einem IP-Datagramm) in einen einzelnen Rahmen der Sicherungsschicht passt. Häufig benutzte Werte für die MTU sind 1.460 Byte, 536 Byte und 512 Byte. Es gab auch Vorschläge, wie die Pfad-MTU bestimmt werden könnte – der größte Rahmen der Sicherungsschicht, der über alle Verbindungen zwischen Quelle und Ziel versendet werden kann [RFC 1191] – und wie die *MSS* basierend auf dieser Pfad-MTU gewählt werden kann. Beachten Sie, dass die *MSS* die maximale Menge an Anwendungsschichtdaten pro Segment ist, nicht aber die Maximalgröße des TCP-Segmentes einschließlich der Header. (Diese Terminologie ist verwirrend, wir müssen aber mit ihr leben, da sie sich eingebürgert hat.)

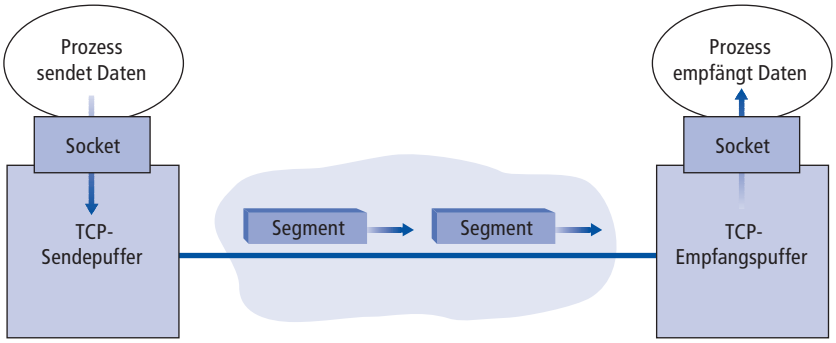


Abbildung 3.28: Puffer im TCP-Sender und -Empfänger

TCP ergänzt jeden Block von Client-Daten um einen TCP-Header und bildet dadurch **TCP-Segmente**. Die Segmente werden zur Netzwerkschicht hinuntergereicht, in der sie dann in Netzwerkschicht-IP-Datagramme verkapselt werden. Diese IP-Datagramme werden dann ins Netz gesandt. Sobald TCP am anderen Ende der Verbindung ein Segment erhält, werden die Daten des Segmentes in den Eingangspuffer der zugehörigen TCP-Verbindung eingefügt, wie ▶Abbildung 3.28 zeigt. Die Anwendung liest den Datenstrom aus diesem Puffer. Jede Seite der Verbindung besitzt ihren eigenen Sende- und ihren eigenen Empfangspuffer. (Betrachten Sie das Applet zur Online-Flusskontrolle auf *http://www.awl.com/kurose-ross*, das eine Animation der Sende- und Empfangspuffer enthält.)

Wir ersehen aus dieser Diskussion, dass eine TCP-Verbindung aus zwei Hälften besteht: Puffer, Variablen und eine Socket-Verbindung zu einem Prozess in einem Host sowie einem weiteren Satz dieser Elemente im anderen Host. Wie bereits erwähnt, sind keinerlei Puffer oder Variablen in den Komponenten im Inneren des Netzwerkes (den Routern und Switches der Sicherungsschicht) mit der Verbindung zwischen den Hosts assoziiert.

3.5.2 TCP-Segmentstruktur

Nachdem wir einen Blick auf die TCP-Verbindung geworfen haben, wollen wir nun die Struktur der TCP-Segmente betrachten. Das TCP-Segment besteht aus Header-Feldern und einem Datenfeld. Letzteres enthält einen Teil der Anwendungsdaten. Wie eben erwähnt, begrenzt die *MSS* die Maximalgröße des Datenfelds eines Segments. Wenn TCP eine große Datei, etwa eine Abbildung auf einer Webseite, sendet, zerlegt es normalerweise die Datei in Stücke der Größe *MSS* (außer dem letzten Stück, das in der Regel kleiner als die *MSS* ist). Interaktive Anwendungen senden jedoch oft Datenstücke, die kleiner als die *MSS* sind; zum Beispiel enthält beim Remote-Login mit Telnet das Datenfeld im TCP-Segment oft nur ein Byte. Weil der TCP-Header normalerweise 20 Byte groß ist (12 Byte mehr als der UDP-Header), sind von Telnet verschickte Segmente durchaus manchmal nur 21 Byte groß.

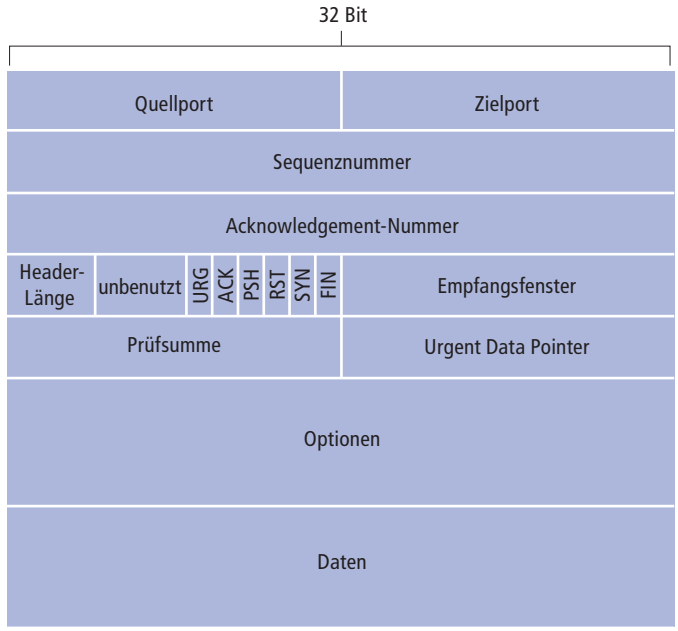


Abbildung 3.29: TCP-Segmentstruktur

▶Abbildung 3.29 zeigt die Struktur des TCP-Segmentes. Wie bei UDP beinhaltet der Header **Quell- und Zielportnummern**, die für das Multiplexing/Demultiplexing der Daten von/zu Anwendungen der nächsthöheren Schicht verwendet werden. Ebenfalls wie bei UDP enthält der Header ein **Prüfsummenfeld**. Ein TCP-Segment-Header enthält zudem die folgenden Felder:

- Das 32-Bit-**Sequenznummernfeld** *(sequence number field)* und das 32-Bit-**Acknowledgment-Nummern-Feld** (*acknowledgment number field)* werden vom TCP-Sender und Empfänger verwendet, um den – später diskutierten – zuverlässigen Datentransferdienst zu realisieren.
- Das 16-Bit-**Empfangsfenster-Feld** *(receive window field)* wird für die Flusskontrolle verwendet. Wir sehen bald, dass es benutzt wird, um dem Sender die Anzahl von Bytes mitzuteilen, die ein Empfänger zu akzeptieren bereit ist.
- Das 4-Bit-**Header-Längenfeld** *(header length field)* gibt die Länge des TCP-Headers in 32-Bit-Worten an. Der TCP-Header kann aufgrund des TCP-Options-Feldes unterschiedlich lang sein. (Normalerweise ist das Optionsfeld leer, so dass die Länge des typischen TCP-Headers 20 Byte beträgt.)
- Die Länge des **Optionsfelds** *(options field)* ist, wie eben bereits erwähnt, variabel. Dieses Feld wird benötigt, wenn Absender und Empfänger die zu verwendende maximale Segmentgröße *(MSS)* aushandeln, aber z.B. auch für einen Faktor zur Fensterskalierung in Hochgeschwindigkeitsnetzen. Außerdem gibt es eine Option für Zeitstempel. In RFC 854 und RFC 1323 finden Sie zusätzliche Details.

- Das **Flag-Feld** *(flag field)* umfasst 6 Bit. Das **ACK-Bit** gibt an, dass der im Acknowledgment-Feld eingetragene Wert gültig ist, d.h., das Segment enthält eine Bestätigung für ein Segment, das erfolgreich empfangen worden ist. Die **RST**-, **SYN-** und **FIN-Bits** werden für den Auf- und Abbau der Verbindung benutzt, wie wir am Ende dieses Abschnittes noch sehen werden. Ein gesetztes **PSH-Bit** zeigt an, dass der Empfänger die Daten sofort an die Anwendungsschicht weiterreichen soll. Schließlich zeigt noch das **URG-Bit**, dass in diesem Segment Daten vorliegen, welche die Instanz der Anwendungsschicht der sendenden Seite als „dringend" gekennzeichnet hat. Die Position des letzten Bytes dieser dringenden Daten wird durch das 16 Bit lange **Urgent-Data-Pointer-Feld** *(Zeiger auf dringende Daten)* gekennzeichnet. TCP muss die Instanz der Empfängerseite über das Vorliegen dringender Daten informieren und ihr einen Hinweis auf das Ende der dringenden Daten liefern. (In der Praxis werden das PSH- und das URG-Flag sowie der Urgent Data Pointer nicht verwendet. Wir erwähnen diese Felder nur der Vollständigkeit halber.)

Sequenznummern und Acknowledgment-Nummern

Zwei der wichtigsten Felder im TCP-Segment-Header sind das Sequenznummernfeld und das Feld für die Acknowledgment-Nummer. Diese Felder sind ein kritischer Teil des zuverlässigen Datentransferdienstes von TCP. Aber bevor wir diskutieren, wie diese Felder für den zuverlässigen Datentransfer verwendet werden, lassen Sie uns zunächst erklären, was TCP eigentlich in diese Felder einträgt.

TCP betrachtet Daten als einen unstrukturierten, aber geordneten Strom von Bytes. Die Verwendung von Sequenznummern durch TCP spiegelt diese Betrachtungsweise wider, denn die Sequenznummern nummerieren den Strom der gesendeten Bytes und nicht die Folge der gesendeten Segmente. Die **Sequenznummer eines Segmentes** ist deshalb die Position des ersten Bytes des Segmentes im Bytestrom. Betrachten wir ein Beispiel. Nehmen wir an, dass ein Prozess in Host A einen Datenstrom an einen Prozess in Host B über eine TCP-Verbindung senden will. TCP in Host A nummeriert implizit jedes Byte im Datenstrom. Nehmen wir weiter an, dass der Datenstrom aus einer Datei der Länge 500.000 Byte besteht, dass die *MSS* 1.000 Byte beträgt und dass das erste Byte des Datenstroms die Nummer 0 hat. Wie ▶Abbildung 3.30 zeigt, erzeugt TCP 500 Segmente aus dem Datenstrom. Das erste Segment erhält die Sequenznummer 0 zugeteilt, das zweite Segment die Sequenznummer 1.000, das dritte Segment die Sequenznummer 2.000 usw. Jede Sequenznummer steht im Sequenznummernfeld im Kopf des entsprechenden TCP-Segmentes.

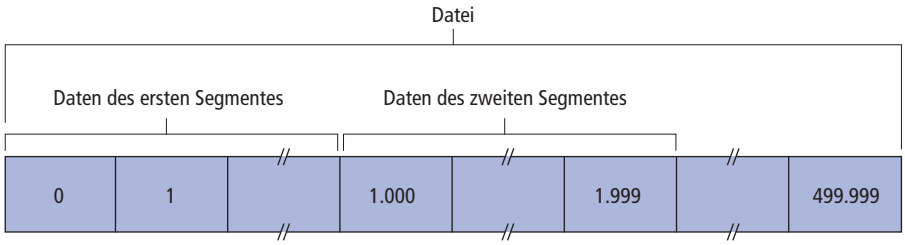


Abbildung 3.30: Das Aufteilen von Daten in TCP-Segmente

Betrachten wir nun die Acknowledgment-Nummern. Sie sind etwas komplexer als die Sequenznummern. Erinnern Sie sich daran, dass TCP Vollduplex bietet, so dass Host A Daten von Host B erhalten kann, während er gleichzeitig Daten an Host B sendet (als Teil derselben TCP-Verbindung). Jedes der Segmente, die von Host B ankommen, hat eine Sequenznummer für die von B zu A fließenden Daten. *Die Acknowledgment-Nummer, die Host A für sein Segment verwendet, ist die Sequenznummer des nächsten Bytes, das Host A von Host B erwartet.* Ein Blick auf einige Beispiele hilft beim Verständnis der Abläufe. Nehmen Sie an, dass Host A alle Bytes von 0 bis 535 erhalten hat und dabei ist, ein Segment an Host B zu senden. Host A wartet auf Byte 536 und alle nachfolgenden Bytes im Datenstrom von Host B. So schreibt Host A die 536 in das Acknowledgment-Nummernfeld des Segmentes, das er an B schickt.

Als anderes Beispiel nehmen Sie an, dass Host A ein Segment von Host B erhält, das die Bytes 0 bis 535 enthält, während ein anderes Segment die Bytes 900 bis 1.000 enthält. Aus irgendeinem Grund hat Host A die Bytes 536 bis 899 noch nicht erhalten. In diesem Beispiel wartet Host A immer noch auf Byte 536 (und folgende), um den Datenstrom von B wieder zusammensetzen zu können. Daher wird das nächste Segment von A an B die Zahl 536 im Acknowledgment-Nummernfeld enthalten. Weil TCP nur Bytes bis zum ersten fehlenden Byte im Strom bestätigt, heißt es, dass TCP **kumulative Acknowledgments** *(cumulative acknowledgments)* verwendet.

Dieses letzte Beispiel bringt auch ein wichtiges, aber subtiles Thema aufs Tablett. Host A hat das dritte Segment (Bytes 900 bis 1.000) vor dem Erhalten des zweiten Segmentes (Bytes 536 bis 899) empfangen. Auf diese Art kam das dritte Segment außer der Reihe an. Das Schwierige daran: Was tut ein Host, wenn er für eine TCP-Verbindung Segmente außer der Reihe erhält? Interessanterweise machen die RFCs von TCP hierfür keine Vorgaben und überlassen die Entscheidung den Menschen, die TCP implementieren. Grundsätzlich gibt es zwei Wahlmöglichkeiten: Entweder (1) der Empfänger verwirft sofort Segmente, die nicht in der richtigen Reihenfolge eintreffen (was, wie wir früher diskutiert haben, das Empfängerdesign vereinfachen kann) oder (2) der Empfänger speichert die erhaltenen Bytes zwischen und wartet auf die fehlenden Bytes, um die Lücke zu schließen. Eindeutig ist letztere Variante hinsichtlich der Nutzung der Netzwerkbandbreite effizienter. Dies ist auch der in der Praxis verwendete Ansatz.

In Abbildung 3.30 nahmen wir an, dass die anfängliche Sequenznummer 0 sei. In Wahrheit wählen beide Seiten einer TCP-Verbindung zufällig eine Anfangssequenznummer aus. Dies wird getan, um die Wahrscheinlichkeit zu reduzieren, dass ein Segment, das aufgrund einer früheren, bereits beendeten Verbindung zwischen zwei Hosts immer noch im Netz herumgeistert, mit einem gültigen Segment aus einer späteren Verbindung zwischen denselben beiden Hosts verwechselt werden kann [Sunshine 1978].

Telnet: eine Fallstudie für Sequenznummern und Acknowledgment-Nummern

Telnet, definiert in RFC 854, ist ein beliebtes Anwendungsschichtprotokoll für das Einloggen auf entfernten Rechnern. Es läuft über TCP und wurde dafür entworfen, zwischen jedem beliebigen Host-Paar zu funktionieren. Im Gegensatz zu den in Kapitel 2 erörterten Anwendungen für die Übertragung größerer Mengen von Daten

ist Telnet eine interaktive Anwendung. Wir erörtern hier ein Telnet-Beispiel, das gut die TCP-Sequenznummern und Acknowledgment-Nummern verdeutlicht. Wir halten fest, dass viele Benutzer mittlerweile das SSH-Protokoll dem Telnet-Protokoll vorziehen, da die Daten einer Telnet-Verbindung (einschließlich der Kennworte) nicht verschlüsselt werden, wodurch Telnet anfälliger für Angriffe ist (wie in Abschnitt 8.7 diskutiert wird).

Nehmen Sie an, dass Host A eine Telnet-Sitzung mit Host B aufbaut. Weil Host A die Sitzung initiiert, wird er als Client bezeichnet und Host B als Server. Jedes vom Benutzer (am Client) eingetippte Zeichen wird dem entfernten Host zugesandt. Dieser schickt eine Kopie jedes Zeichens zurück, das auf dem Bildschirm des Telnet-Benutzers angezeigt wird. Dieses „echo back“ *(zurückgesandtes Echo)* stellt sicher, dass vom Telnet-Benutzer gesehene Zeichen auch wirklich schon am entfernten Standort empfangen und verarbeitet worden sind. Jedes Zeichen durchquert zwischen dem Zeitpunkt, in dem der Benutzer die Taste anschlägt, und der Zeit, zu dem es auf dem Monitor des Benutzers erscheint, zweimal das Netz.

Nehmen wir nun an, dass ein Benutzer eine einzelne Taste drückt und sich dann einen Kaffee holen geht. Betrachten wir die TCP-Segmente, die zwischen dem Client und dem Server ausgetauscht werden. Wie in ▶Abbildung 3.31 gezeigt nehmen wir an, dass die anfänglichen Sequenznummern 42 und 79 seitens des Clients bzw. des Servers sind. Erinnern Sie sich daran, dass die Sequenznummer eines Segmentes die

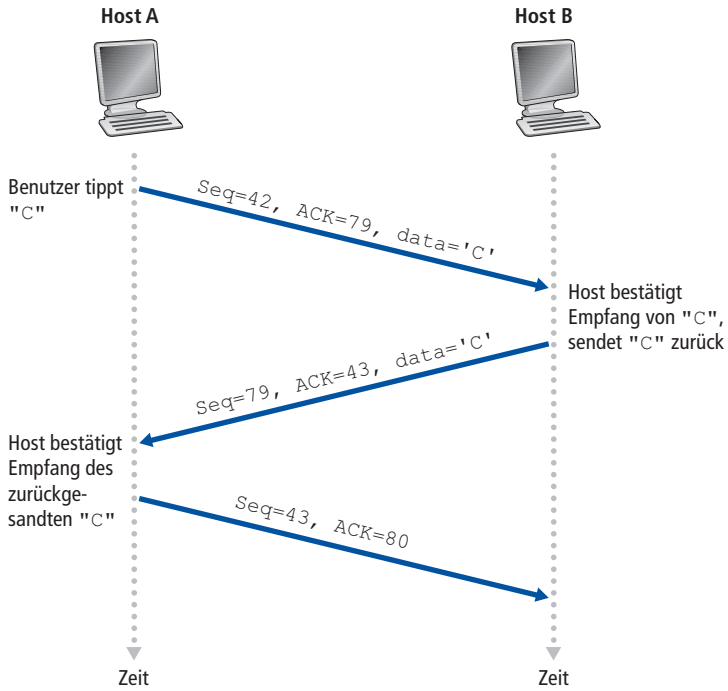


Abbildung 3.31: Sequenznummern und Acknowledgement-Nummern für eine einfache Telnet-Anwendung über TCP

Copyright

Daten, Texte, Design und Grafiken dieses eBooks, sowie die eventuell angebotenen eBook-Zusatzdaten sind urheberrechtlich geschützt. Dieses eBook stellen wir lediglich als **persönliche Einzelplatz-Lizenz** zur Verfügung!

Jede andere Verwendung dieses eBooks oder zugehöriger Materialien und Informationen, einschließlich

- der Reproduktion,
- der Weitergabe,
- des Weitervertriebs,
- der Platzierung im Internet, in Intranets, in Extranets,
- der Veränderung,
- des Weiterverkaufs und
- der Veröffentlichung

bedarf der **schriftlichen Genehmigung** des Verlags. Insbesondere ist die Entfernung oder Änderung des vom Verlag vergebenen Passwortschutzes ausdrücklich untersagt!
Bei Fragen zu diesem Thema wenden Sie sich bitte an: info@pearson.de

Zusatzdaten

Möglicherweise liegt dem gedruckten Buch eine CD-ROM mit Zusatzdaten bei. Die Zurverfügungstellung dieser Daten auf unseren Websites ist eine freiwillige Leistung des Verlags. **Der Rechtsweg ist ausgeschlossen.**

Hinweis

Dieses und viele weitere eBooks können Sie rund um die Uhr und legal auf unserer Website herunterladen:

http://ebooks.pearson.de