

Datenkommunikation

Teil 3.3: Transportschicht (TCP, UDP)

O.Univ.Prof.Dr. Harmen R. van As

Übersicht

3.3 Internet-Referenzmodell: Transportschicht

- TCP (Transmission Control Protocol)
- UDP (User Data Protocol)
- TCP und UDP Formate, Eigenschaften der Protokolle
- TCP Flusskontrolle

Transportprotokolle im Internet

• User Datagram Protocol (UDP)

- verbindungslos
- Nachrichten-orientiert
- Multicast-Unterstützung
- unzuverlässig
 - optionale Fehlererkennung
 - keine Fehlerbehebung
 - keine Reihenfolgeerhaltung

• Transmission Control Protocol (TCP)

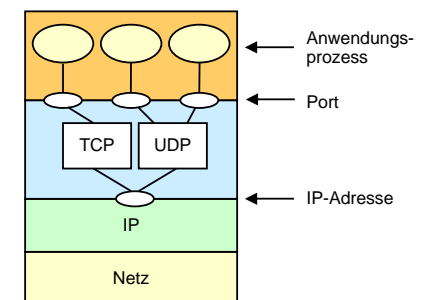
- verbindungsorientiert
- Bytestrom-orientiert
- unterstützt nur 1:1-Kommunikation
- zuverlässig

Transport-Adressen im Internet

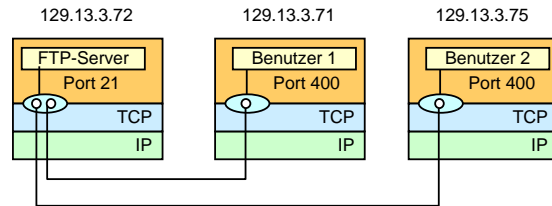
• Ziel:

Kommunikation zwischen Prozessen

- Prozess-ID des Betriebssystems ist zur eindeutigen Identifikation von Prozessen nicht geeignet.
- vollständige Transportadresse = IP-Adresse + Port-Nummer
- Port = Kommunikationsendpunkt



Adressierung



- Identifikation von TCP-Diensten über Ports (vergleichbar mit SAPs)
- reservierte Portnummern bis 255 für häufig benutzte Dienste (well-known ports)
- Socket: IP-Adresse + Port
(vergleichbar mit Verbindungsendpunkt CEP innerhalb von SAP)
- **Beispiel**
 - Verbindung mehrerer Benutzer mit Port-Nummer 400 zu 1 FTP-Server
 - Identifizierung der Verbindungen über IP-Adresse und Port-Nummer

Anwendungsprotokolle mit TCP oder UDP

Anwendung	Anwendungs-protokoll	Verwendetes Transportprotokoll
Email	SMTP	TCP
Remote terminal access	Telnet	TCP
Web	HTTP	TCP
Dateitransfer	FTP	TCP
Entfernter Fileserver	NFS	UDP
Streaming multimedia	Proprietär	UDP
Internet-Telefonie	Proprietär	UDP
Netzmanagement	SNMP	UDP
Intra-Domain-Routing	RIP	UDP
Inter-Domain-Routing	BGP	TCP
Namensübersetzung	DNS	UDP

IPv4 und IPv6 Header

IPv4

Version	IHL	Type of Service	Total Length		
Identification			Flags	Fragment Offset	
Time to Live	Protocol	Header Checksum			
Source Address					32 bit
Destination Address					32 bit
Options				Padding	

IPv6

Version	Priority	Flow Label	
Payload Length		Next Header	Hop Limit
Source Address			
Destination Address			

128 bit

128 bit

Adressierung: Protokoll und Port

IP Protocols Numbers

0		Reserved
1	ICMP	Internet Control Message Protocol
2	IGMP	Internet Group Management Protocol
3	GGP	Gateway-to-Gateway Protocol
4	IP	IP encapsulation
5	TCP	Transmission Control Protocol
8	EGP	Exterior Gateway Protocol
9	IGP	Interior Gateway Protocol
17	UDP	User Datagram Protocol
41	IPv6	IP version 6
50	ESP	Encapsulation Security Payload for IPv6
51	AH	Authentication Header for IPv6
89	OSPF	Open Shortest Path

Well-known Ports

13	NTP	Network Time Protocol
20	FTP	File Transfer Protocol - Daten
21	FTP	File Transfer Protocol - Kontrolldaten
25	SMTP	Simple Mail Transfer Protocol
53	DNS	Domain Name Server
80	HTTP	Hyper Text Transfer Protocol
119	NNTP	Network News Transfer Protocol

Extension Header in IPv6

Base Header

Base Header Next = TCP	TCP Segment
---------------------------	----------------

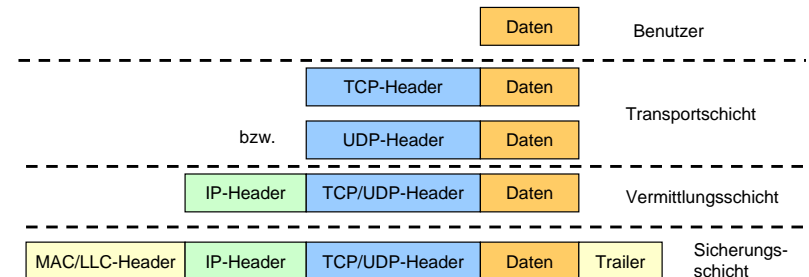
Base Header and One Extension Header

Base Header Next = Route	Route Header Next = TCP	TCP Segment
-----------------------------	----------------------------	----------------

Base Header and Two Extension Headers

Base Header Next = Route	Route Header Next = Auth	Auth Header Next = TCP	TCP Segment
-----------------------------	-----------------------------	---------------------------	----------------

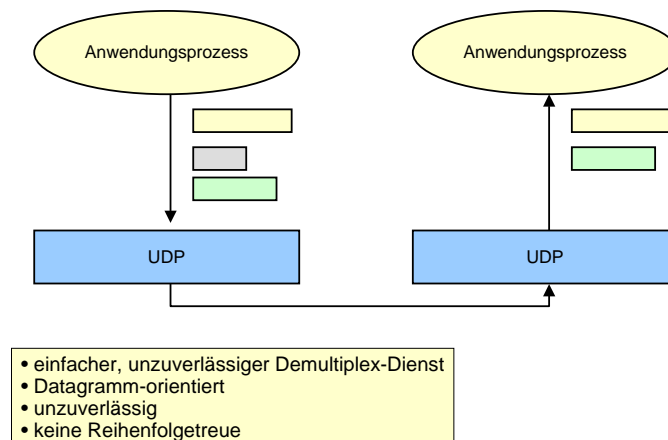
PDU Verschachtelung



Transport- versus Sicherungsprotokolle

Sicherungsprotokolle	Transportprotokolle
Instanzen sind über einen Link verbunden	Instanzen sind in nicht benachbarten Systemen
eher feste, kleine Paketumlaufzeiten (round trip times, RTTs)	stark variierende RTTs, dadurch hohes Bandbreiten- Verzögerungs-Produkt
keine Reihenfolgevertauschungen	Mögliche Reihenfolgevertauschungen
Link limitiert Senderate	möglicher Engpass auf einem Zwischen-Link

User Datagram Protocol (UDP)



UDP-Header

Source Port	Destination Port
Length	Checksum
Daten	

Prüfsummenberechnung

Source IP Address		
Destination IP Address		
0	Protocol	Segment Length
UDP/TCP-Header + Daten		

- Definition eines Pseudo-Headers
- Berechnung der Prüfsumme (optional in UDP über IPv4) über
 - Pseudo-Header
 - TCP/UDP-Header
 - Daten
- Absicherung von IP-Protokollinformation
- erlaubt Schutz gegen fehlgeleitete UDP-Pakete
- Ist Prüfsummenfeld = 0, dann wird keine Prüfsummenberechnung gewünscht
Bei berechneter Prüfsumme 0 wird 0xFFFF übertragen

Eigenschaften von UDP

Keine Verbindungsaufbauphase

Verzögerung bis zum Aufbau einer Verbindung entfällt. Daten können sofort gesendet werden.

Kein Verbindungszustand

- Im Endsystem müssen keine verbindungsrelevanten Informationen gehalten werden (z.B. Flusskontrollfenster, Staukontrollfenster, Sequenznummern)
- Server kann mittels UDP typischerweise mehr aktive Clients unterstützen als mit

Geringer Overhead in der Dateneinheit

- Lediglich Adressen sowie Längenfeld und Prüfsumme

Unreguliertes Senden

UDP kann Daten so schnell senden wie sie von der Anwendung geliefert werden und wie sie vom Netz abgenommen werden (Verlust bei Sender möglich)

Vorteil: Sehr einfaches Protokoll mit äußerst geringem Overhead

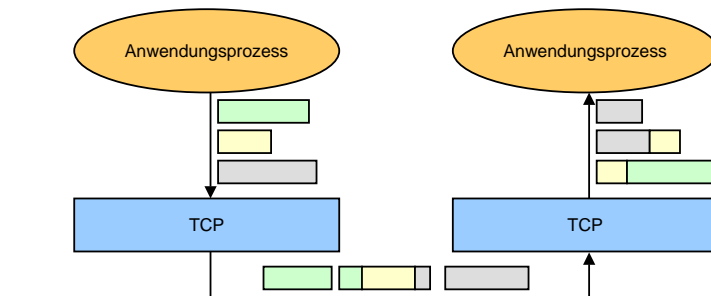
UDP Port Numbers

- < 512: reserved for particular services (one per host)
- 512 ... 1023: privileged port (Unix superuser only)
- 1024 ... 65535: available for applications
- some services offered both as UDP and TCP *may* have same port number
 - 7 echo
 - 9 discard
 - 19 character generator
 - 37 time
 - 53 domain name service
 - 123 Network Time Protocol (NTP)

Transmission Control Protocol (TCP)

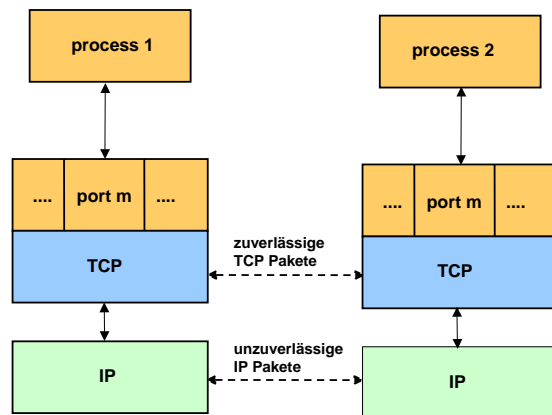
- Verbindungsaufbau zwischen zwei Sockets (Verbindungsendpunkte)
- Datenübertragung
 - Vollduplex über virtuelle Verbindung
 - Unterstützung von Prioritäten
 - Reihenfolgetreue
 - Fluss-/Staukontrolle mit Fenstermechanismus
 - Fehlerkontrolle
 - Folgenummern
 - Prüfsumme
 - Quittierungsnummern
 - Übertragungswiederholung
- Gesicherter Verbindungsabbau

TCP-Datenaustausch

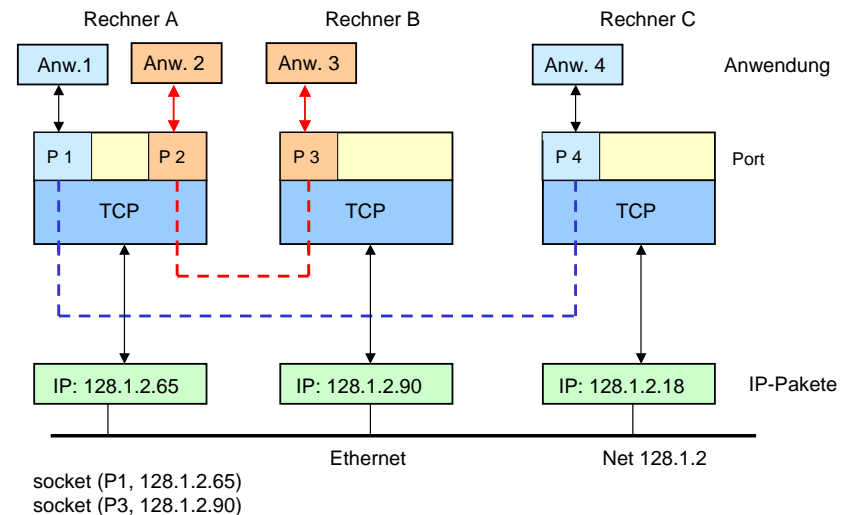


- unterstützt reihenfolgetreuen und zuverlässigen Byte-Strom
- Segmentgröße richtet sich nach MTU-Größe des lokalen Links oder Segment wird nach Push-Operation bzw. Timeout erzeugt

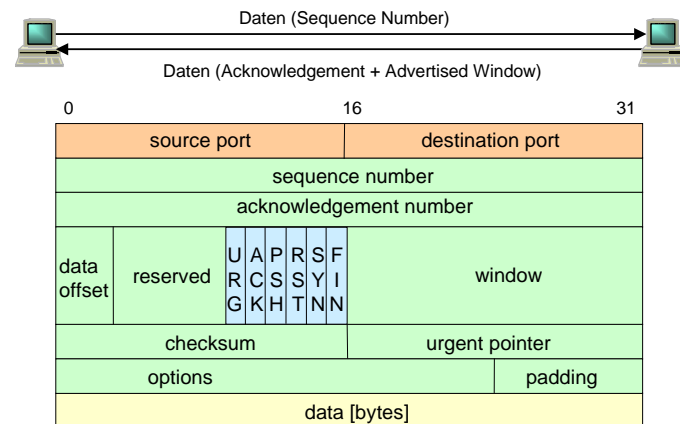
TCP Verbindung



TCP Verbindungen



TCP Header



URG Urgent pointer field significant RST Reset the connection
 ACK Acknowledge field significant SYN Synchronize the sequence numbers
 PSH Push function FIN Finalize, no more data from sender

Bedeutung der Felder eines TCP-Header

Quell-Port und Ziel-Port: identifizieren die Endpunkte der Verbindung. Freie Zuweisung der Ports in den Endsystemen (bis auf die standardisierten Nummern bis 255).

Sequenznummer: Sequenznummer gemessen in Byte.

Quittung: Die nächste vom Empfänger erwartete Sequenznummer.

Offset: Anzahl der 32-Bit-Wörter im TCP-Kopf.

URG: Wird auf 1 gesetzt, falls der Urgent Pointer verwendet wird. Wird i.d.R. nicht benutzt.

SYN: Wird beim Verbindungsaufbau verwendet, um CONNECT.Req/Ind anzuzeigen.

ACK: Unterscheidet bei gesetztem SYN-Bit eine CR-PDU von einer CC-PDU. Signalisiert die Gültigkeit des Acknowledgement-Feldes.

FIN: Gibt an, dass der Sender keine Daten mehr senden möchte.

RST: Wird benutzt, um eine Verbindung zurückzusetzen.

PSH: Signalisiert, dass die übergebenen Daten sofort weitergeleitet werden sollen. Gilt sowohl für den Sender als auch für den Empfänger. Wird fast nicht benutzt.

Empfangsfenster: Dient zur Flusskontrolle. Anzahl der Bytes, die nach dem höchsten bestätigten Byte gesendet werden dürfen.

Prüfsumme: Enthält die Prüfsumme über TCP-Kopf und Daten. Wie bei UDP.

Urgent-Zeiger: Relativer Zeiger auf wichtige Daten.

Optionen-Feld: kann Optionen variabler Länge aufnehmen (n x 32 Bit).

Well Known Ports

20	FTP (Data), File Transfer Protocol	(TCP)
21	FTP (Control)	(TCP)
23	TELNET, Terminal Emulation	(TCP)
25	SMTP, Simple Mail Transfer Protocol	(TCP)
53	DOMAIN, Domain Name Server	(UDP)
67	BOOTPS, Bootstrap Protocol Server	(UDP)
68	BOOTPC, Bootstrap Protocol Client	(UDP)
69	TFTP, Trivial File Transfer Protocol	(UDP)
80	HTTP Hypertext Transfer Protocol (default port)	(TCP)
111	SUN RPC, Run Remote Procedure Call	(TCP)
161	SNMP, Simple Network Management Protocol	(UDP)

TCP-Verbindungsaufbau

- Aufbau von Verbindungen nach Erzeugen eines Sockets

– **Aktiver Modus:** Anforderung einer TCP-Verbindung mit spezifiziertem Socket (connect)

– **Passiver Modus:** Benutzer informiert TCP, dass er auf eingehende Verbindung wartet (listen/accept)

- Alternativen:
 - spezieller Socket
 - Annahme aller Verbindungen

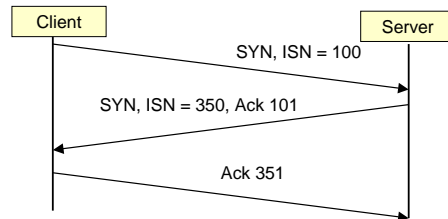
TCP Mechanism

Connection Establishment

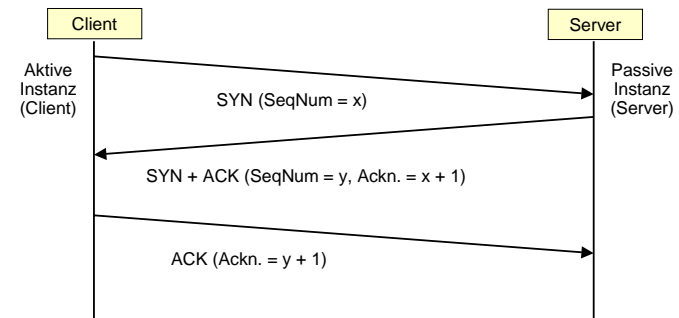
- Three-way handshake
- SYN flag set \Rightarrow Request for connection

Connection Termination

- Close with FIN flag set
- Abort

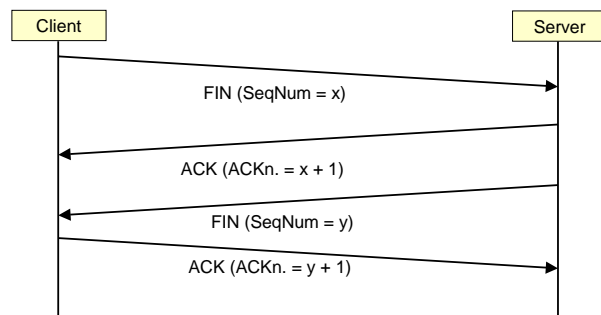


Öffnen einer Verbindung



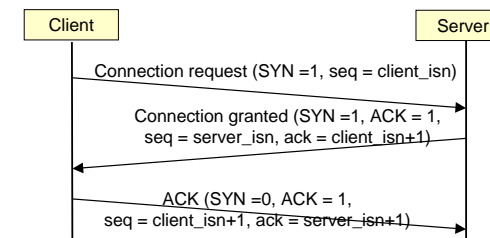
- 3-Wege-Handshake zum Verbindungsaufbau
- Austausch von Start-Sequenznummern
(Möglicherweise wurde Verbindung mit gleichen Port-Nummern kürzlich beendet und es befinden sich noch Pakete im Netz.)

Gesichertes Beenden einer TCP-Verbindung



- Beide Seiten der Verbindung werden geschlossen
- Verfahren stellt sicher, dass alle gesendeten Daten vor Beenden der Verbindung ankommen

TCP-Verbindungsverwaltung



Verbindungsaufbau

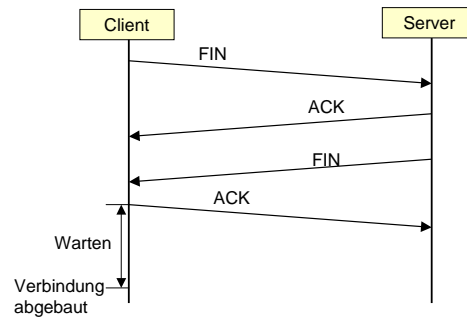
- Unterscheidung: Client (initiiert Verbindung) und Server (wartet auf Verbindungswünsche)
- Connection request und connection granted führen keine Nutzdaten mit sich
 - Festlegung der initialen Sequenznummern (..._isn)
 - Aushandlung der Größe des Flusskontrollfensters
 - Allokation der Puffer
- Einige Optionen sind nur beim Verbindungsaufbau erlaubt bzw. ausgehandelt (z.B. WindowScaling)

isn: initial sequence number

TCP-Verbindungsverwaltung

Verbindungsabbau

- Kann sowohl vom Client als auch vom Server initiiert werden
- Weitere Varianten für simultanen Verbindungsabbau existieren



Verbindungsabbau

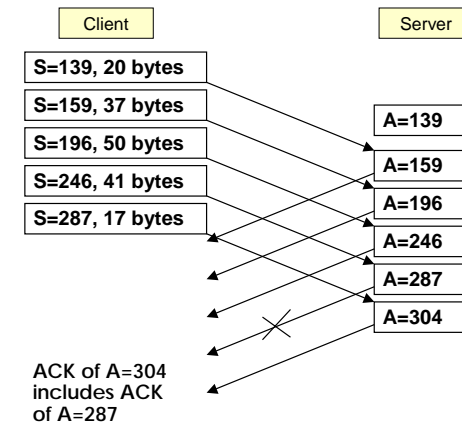
Ordnungsgemäßer Abbau

- Beide Anwendungsprozesse müssen unabhängig voneinander ihre "Hälften" der Verbindung schließen
- wurde lediglich eine Richtung geschlossen, so können in der anderen Richtung noch Daten gesendet werden. Damit ist noch ein unidirektionaler Datenfluss möglich.
- 3-Wege-Handshake

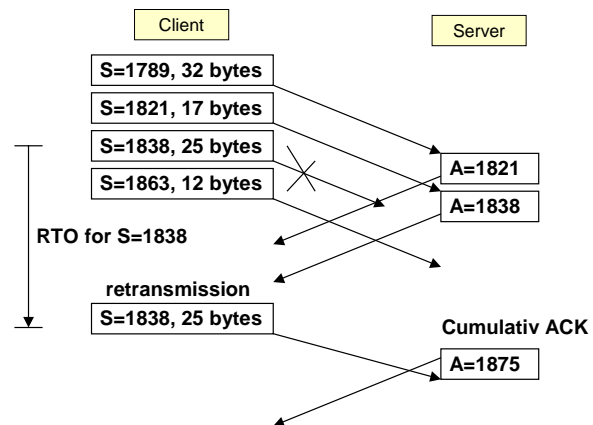
Abbruch einer Verbindung

- Verbindung kann mit Reset abgebrochen werden (RST-Segment)

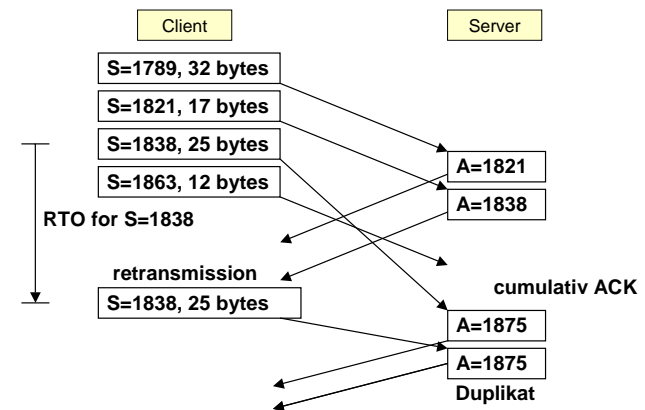
Advantage of Cumulative Acknowledgement



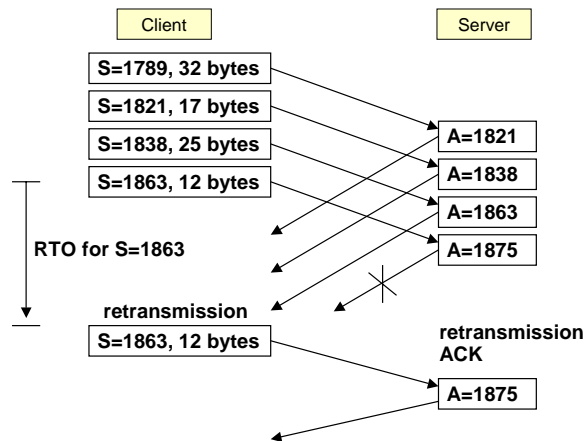
Retransmission / Original Segment lost



Retransmission / Delay too long

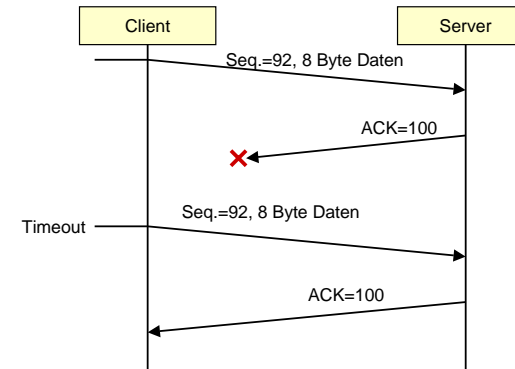


Retransmission / Acknowledgement lost



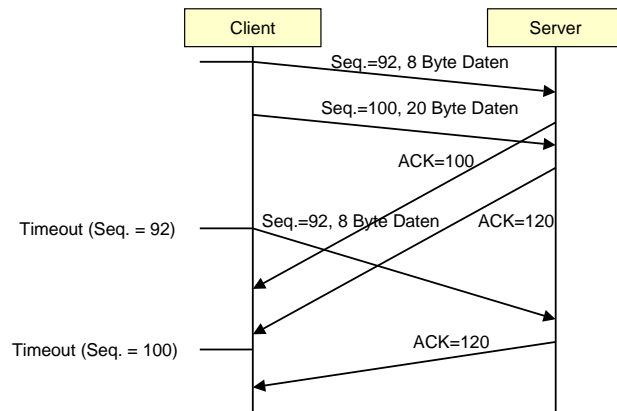
Sendewiederholungen: Szenario 1

Sendewiederholung wegen verlorener Quittung



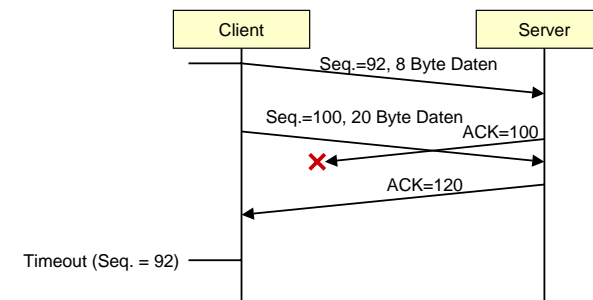
Sendewiederholungen: Szenario 2

Sendewiederholung aufeinanderfolgender Dateneinheiten

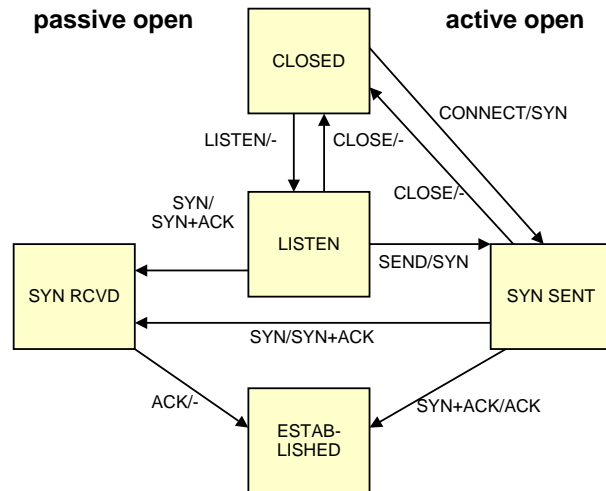


Sendewiederholungen: Szenario 3

Kumulative Quittung vermeidet Sendewiederholung
Erweiterungen der Quittierung in TCP
- RFC 1818: Selektive Quittungen



TCP-Automat bei Verbindungsaufbau



TCP (Transmission Control Protocol)

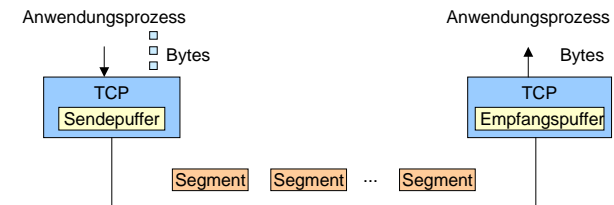
Stellt zuverlässigen, verbindungsorientierten Punkt-zu-Punkt-Dienst zur Verfügung
Arbeitet an der Dienstschnittstelle mit einem Bytestrom
(RFC 793, September 1981)

Ist in der Regel über IP implementiert

Die zu IP weitergereichte Dateneinheit wird als TCP-Segment bezeichnet
Logische Sicht auf TCP-Ebene

Neue / optimierte Funktionen

- Verbesserung des Timeout-Mechanismus (Zeitstempel im Segment-Kopf)
- Erhöhen der Fenstergröße (Basiseinheit kann skaliert werden)

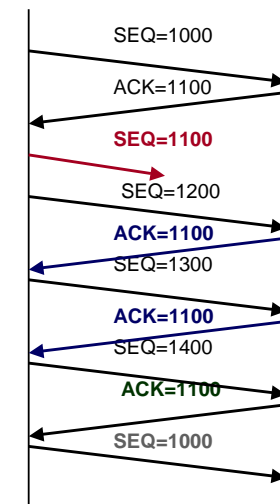


Fehlerbehebung

- Quittierungsstrategien
 - sofort
 - kumulativ
 - optional: selektiv
- Übertragungswiederholung nach ausbleibender Quittung
 - Standard: Go-Back-N
 - optional: selektive Übertragungswiederholung

Fast Retransmit

- **Problem:**
 - möglicherweise lange Wartezeit auf Übertragungswiederholung durch lange Timeouts
- **Ansatz:**
 - Jedes Paket wird quittiert
 - Bei einem verlorenen Paket erzeugt das folgende Paket eine Quittung mit unverändertem Acknowledgment-Wert
→ **Duplicate ACK**
 - Übertragungswiederholung nach **3. Duplicate ACK**



TCP Flusskontrolle

- Flusskontrolle regelt den Datenfluss zwischen Endsystemen.
- Flusskontrolle in TCP mit Fenstermechanismus
 - Acknowledgment-Feld bestätigt Empfang aller niedrigeren Sequenznummern.
 - AdvertisedWindow-Feld gibt an, wieviele Bytes der Empfänger zusätzlich akzeptiert.
 - Empfänger erlaubt dem Sender das Senden von Daten bis Acknowledgment + AdvertisedWindow.

Sequenznummernüberlauf

Bitrate	Zeit bis Sequenznummernüberlauf
T1 (1.5 Mbit/s)	6.4 h
Ethernet (10 Mbit/s)	57 min
FDDI (100 Mbit/s)	6 min
STM-1 (155 Mbit/s)	4 min
STM-3 (622 Mbit/s)	55s
STM-16 (2.49 Gbit/s)	14s

← MSL = 2 min

MSL: maximale Segment-Lebenszeit

Bitrate-Verzögerungs-Produkt

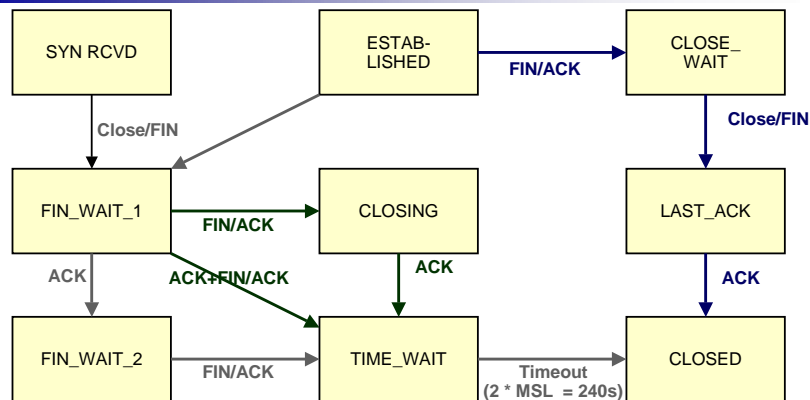
Bitrate	Bitrate x Verzögerung (RTT = 100 ms)
T1 (1.5 Mbit/s)	18 KB
Ethernet (10 Mbit/s)	122 KB
FDDI (100 Mbit/s)	549 KB
STM-1 (155 Mbit/s)	1.2 MB
STM-3 (622 Mbit/s)	1.8 MB
STM-16 (2.5 Gbit/s)	29.8 MB

- **Probleme**
 - kein effizienter kontinuierlicher Datenfluss
 - Sequenznummernbereich läuft schnell über
 - Advertised Window (16 Bit) erlaubt Fenster von 64 KB

TCP-Optimierungen

- **RTT-Bestimmung**
 - Übertragen eines 32-Bit-Timestamp in TCP-Option
 - Empfänger sendet Timestamp in Quittung zurück.
- **Schutz vor zu frühem Sequenznummernüberlauf**
 - Erweiterung der Sequenznummer um Timestamp zur Erkennung eines Sequenznummernüberlaufs
 - 64 Bit Identifikator des Pakets
 - Timestamp in den höherwertigen 32 Bits
 - Sequenznummer in den niederwertigen 32 Bits
- **Vergrößern des Sendefensters bei großem Bitrate-Verzögerungs-Produkt**
 - Aushandeln eines Skalierungsfaktors für AdvertisedWindow

TCP-Automat bei Verbindungsabbau



Gegenseite schließt Verbindung zuerst
Eigene Seite schließt Verbindung zuerst
Gleichzeitiges Schließen beider Seiten

Senden einer TCP-Dateneinheit

Empfang von der Anwendung: Bytestrom
Senden an IP: TCP-Dateneinheiten (TCP-Segemente)

Problem

- Wann wird aus dem empfangenen Bytestrom eine TCP-Dateneinheit an IP weitergegeben?
Nach RFC 793 (TCP): „send that data in segments at its own convenience“

Alternativen

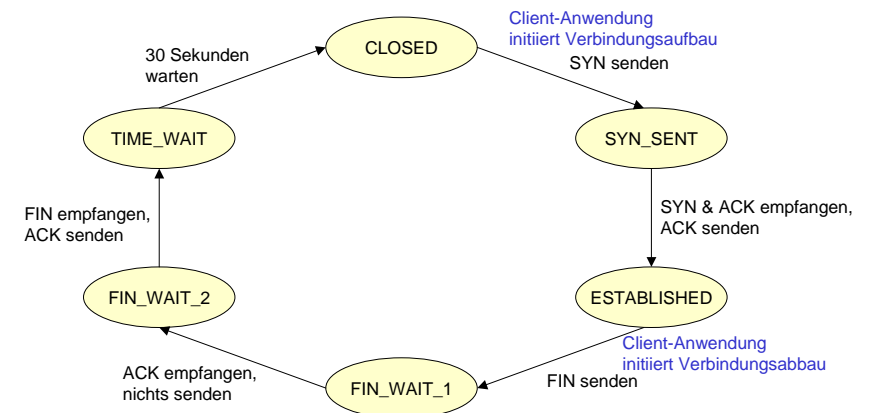
- MSS: Maximum Segment Size
kann sich an der Größe der maximalen Dateneinheit des direkt angeschlossenen Netzes orientieren, an der eventuell ermittelten Maximum Transfer Unit (MTU) oder an der Segmentierungsgröße, mit der IP umgehen können muss.
- Längenangabe der Anwendungsdaten an, nicht Länge der TCP-Dateneinheit
- Typische Größen (vermeiden das Segmentieren durch IP), 1460 Byte, 536 Byte, 512 Byte
- Push (PSH-Flag im Kopf der TCP-Dateneinheit)
Sender verlangt hiermit das sofortige Versenden der übergebenen Daten (Vorrangdaten) wird beispielsweise bei Telnet verwendet
- Zeitgeber
nach einem gewissen Zeitintervall der Inaktivität werden vorhandene Daten gesendet

Key Features

- Connection oriented
- Point-to-point communication: Two end-points
- Reliable transfer: Data is delivered in order
- Full duplex communication
- Stream interface: Continuous sequence of octets
- Reliable connection startup: Data on old connection does not confuse new connections
- Graceful connection shutdown: Data sent before closing a connection is not lost.

Zustandsübergangsdiagramm TCP-Client

Typische Sequenz von Zuständen



Es handelt sich hier um eine vereinfachte Darstellung und nicht um den TCP-Zustandsautomaten

Generierung von TCP-Quittungen

Ereignis	Reaktion des TCP-Empfängers
Ankunft einer Dateneinheit in Reihenfolge. Alle Daten davor bereits quittiert. Keine Lücken.	Verzögerte Quittung. Bis zu 500 ms warten auf weitere reihenfolgetreue Dateneinheit. Wird keine weitere empfangen, dann Senden der Quittung.
Ankunft einer Dateneinheit in Reihenfolge. Alle Daten davor bereits quittiert. Bereits eine weitere reihenfolgetreue auf Quittung wartende Dateneinheit. Keine Lücken.	Sofortiges Senden einer kumulativen Quittung. Beide Dateneinheiten werden quittiert.
Ankunft einer Dateneinheit mit einer nicht erwarteten höheren Sequenznummer. Erkennen einer Lücke.	Sofortiges Versenden einer duplizierten Quittung mit der Sequenznummer des nächsten erwarteten Bytes.
Ankunft einer Dateneinheit, die teilweise oder komplett eine Lücke bei den empfangenen Daten auffüllt.	Sofortiges Versenden einer Quittung, falls die Dateneinheit an der unteren Schranke der Lücke beginnt.

stehen sendebereite Daten beim Empfänger zur Verfügung, werden Quittungen per Piggyback gesendet

Generierung von TCP-Quittungen

- Sofortiges Versenden einer Quittung, falls die Dateneinheit an der unteren Schranke der Lücke beginnt.
- Ankunft einer Dateneinheit, die teilweise oder komplett eine Lücke bei den empfangenen Daten auffüllt.
- Sofortiges Versenden einer duplizierten Quittung mit der Sequenznummer des nächsten erwarteten Bytes.
- Ankunft einer Dateneinheit mit einer nicht erwarteten höheren Sequenznummer.
- Erkennen einer Lücke. Sofortiges Senden einer kumulativen Quittung.
- Beide Dateneinheiten werden quittiert. Ankunft einer Dateneinheit in Reihenfolge.
- Alle Daten davor bereits quittiert. Bereits eine weitere reihenfolgetreue auf Quittung wartende Dateneinheit. Keine Lücken. Verzögerte Quittung. Bis zu 500 ms warten auf weitere reihenfolgetreue Dateneinheit.
- Wird keine weitere empfangen, dann Senden der Quittung.
- Ankunft einer Dateneinheit in Reihenfolge. Alle Daten davor bereits quittiert.
- Keine Lücken.

Reaktion des TCP-Empfängers Ereignis

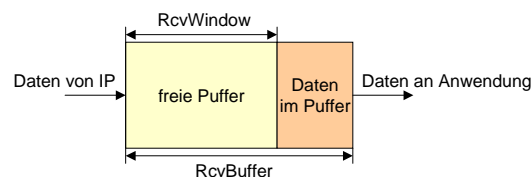
- stehen sendebereite Daten beim Empfänger zur Verfügung,
- werden Quittungen per Piggyback gesendet.

TCP-Flusskontrolle

Empfangsfenster

- gibt an, wieviel Pufferplatz der Empfänger für diese Verbindung zur Verfügung hat (explizite Kreditvergabe=
- Feld Empfangsfenster im Kopf der TCP-Dateneinheit
- RcvWindow
- kann dynamisch geändert werden

Schema beim Empfänger



TCP-Flusskontrolle

Szenario: Endsystem A schickt große Datei über TCP an Endsystem B

Variablen beim Empfänger

- **LastByteRead:** letzte Sequenznummer, die von der Anwendung aus dem Empfangspuffer gelesen wurde
- **LastByteRcvd:** letzte Sequenznummer, die über das Netz empfangen und in den Empfangspuffer geschrieben wurde

Es muss immer gelten: $\text{LastByteRcvd} - \text{LastByteRead} \leq \text{RcvBuffer}$

Empfangsfenster: $\text{RcvWindow} = \text{RcvBuffer} - (\text{LastByteRcvd} - \text{LastByteRead})$

Variablen beim Sender

- **LastByteSent:** letzte Sequenznummer, die gesendet wurde
- **LastByteAcked:** letzte Sequenznummer, die quittiert wurde

Es muss immer gelten: $\text{LastByteSent} - \text{LastByteAcked} \leq \text{RcvWindow}$

Zero Window Probing

Problem: Fenstergröße von Null

- Endsystem B hat an Endsystem A gemeldet, dass sein Empfangspuffer voll ist, damit ist das Empfangsfenster leer, d.h. RcvWindow=0
- Endsystem A kann keine Daten mehr senden.
 - Konsequenz, Endsystem B kann Endsystem A nicht darüber informieren, dass wieder freier Empfangspuffer existiert
- Forderung
 - Endsystem A muss auch bei RcvWindow=0 Dateneinheiten mit 1 Byte senden, die von Endsystem B quittiert werden
 - Persistent-Timer steuert das Zero Window Probing

Umlaufzeit und Timeout

Problem

- Auf welchen Wert soll der TCP-Timeout gesetzt werden?

Beobachtungen

- Wert sollte sich an der Umlaufzeit (Round Trip Time, RTT) orientieren und etwas größer sein

Umlaufzeit variiert

- zu kurzer Wert: unnötige Sendewiederholungen treten auf
- zu langer Wert: langsame Reaktion auf Verlust der Dateneinheit

Bestimmung der Umlaufzeit

Messung der Umlaufzeit

Timer (Granularität variiert, bis zu 500 ms) wird benutzt. Beim Ablauf wird ein Zähler jeweils inkrementiert.

SampleRTT

- Gemessene Zeit vom Versenden der Dateneinheit bis zum Empfang der dazugehörigen Quittung.
- Sendewiederholungen werden ignoriert.
- Kumulativ quittierte TCP-Dateneinheiten werden nicht betrachtet.

Glätten des gemessenen Wertes

- Gemessener Wert kann stark schwanken
- Es wird ein auf mehreren Messungen basierender Wert herangezogen: **EstimatedRTT**

Bestimmung der Umlaufzeit

Nach jedem erfassten Wert für SampleRTT wird der geglättete Wert der Umlaufzeit folgendermaßen bestimmt

$$\text{EstimatedRTT} = (1-a) * \text{EstimatedRTT} + a * \text{SampleRTT}$$

- Exponential Weighted Moving Average (EWMA)
- Neue Werte werden höher gewichtet als alte Werte
- Einfluss eines gemessenen Wertes sinkt exponentiell
- Typischer Wert für a: 0.125
- $\text{EstimatedRTT} = 0.875 \text{ EstimatedRTT} + 0.125 \text{ SampleRTT}$

Bestimmen des Timeout-Werts

Sollte auf einen etwas höheren Wert gesetzt werden als EstimatedRTT

Bei hohen Schwankungen von EstimatedRTT sollte ein höherer Sicherheitszuschlag gegeben werden

$\text{Timeout} = \text{EstimatedRTT} + 4 * \text{Deviation}$

$\text{Deviation} = (1 - a) * \text{Deviation} + a * |\text{SampleRTT} - \text{EstimatedRTT}|$

Timer in TCP

Neben dem eben besprochenen Retransmission Timer verfügt TCP noch über eine Reihe weiterer Timer:

• Persist Timer

- falls Fenstergröße auf Null ist und Quittungen verloren gehen, kann es zu einem Deadlock kommen
- Persist-Timer initiiert regelmäßige Nachfragen nach der Fenstergröße, auch wenn der Empfänger sein Empfangsfenster schließt
- TCP exponential backoff zur Berechnung der Timer-Werte

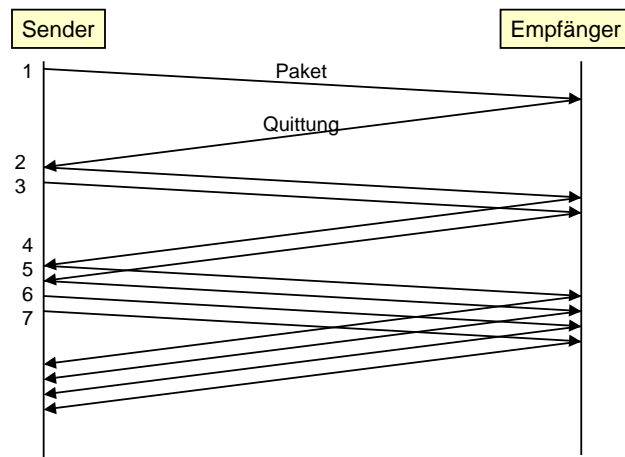
• Keepalive Timer

- Erkennt, wenn der Partner Probleme hat über eine TCP-Verbindung im Idle-Zustand fließen keine Daten
- ist nicht Bestandteil der TCP-Spezifikation, stellt eine Option dar
- kann dazu führen, dass bestehende Verbindungen terminieren (z.B. bei einem temporären Problem auf der Vermittlungsschicht)
- kann Servern helfen, Ressourcen nicht unnötig zu belegen, falls Client abgestürzt ist

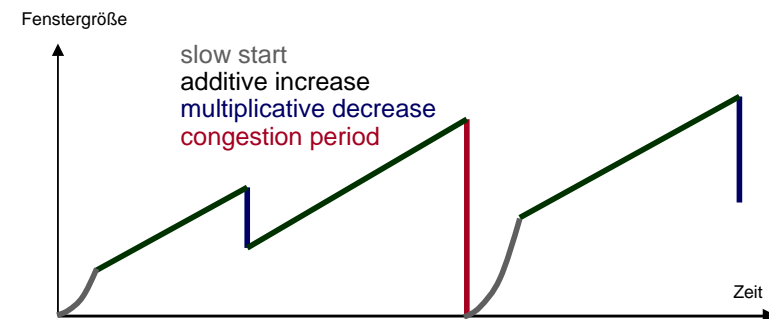
• 2MSL-Timer

- Misst die Zeit, die eine Verbindung im TIME_WAIT-Zustand verbringt

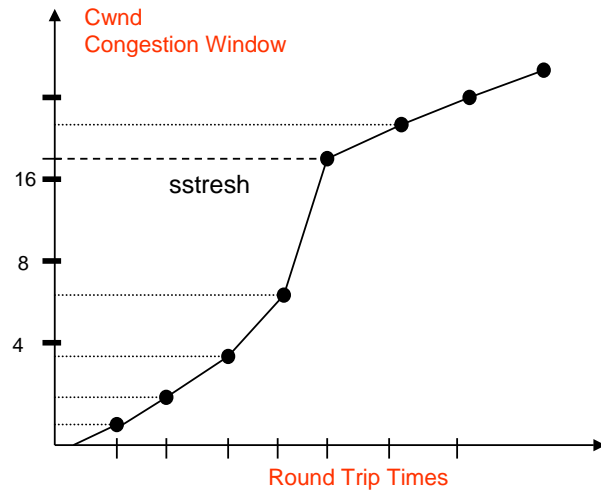
TCP Slow Start



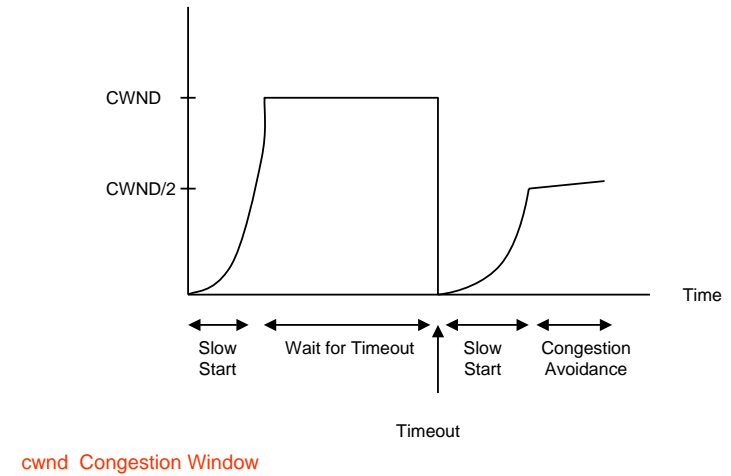
Slow Start



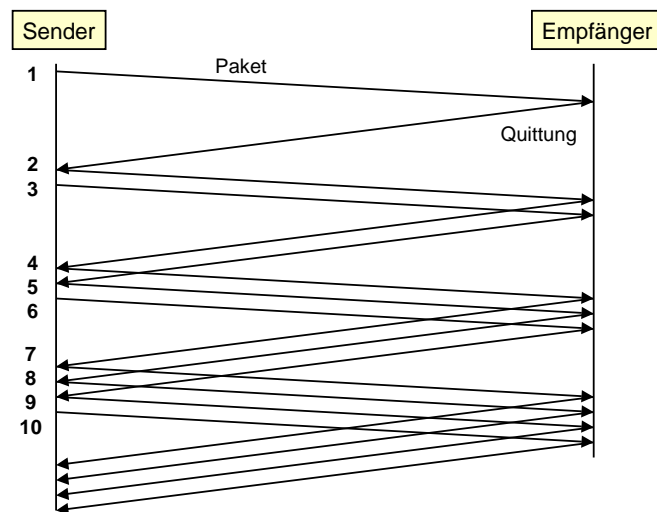
TCP Slow Start and Congestion Avoidance



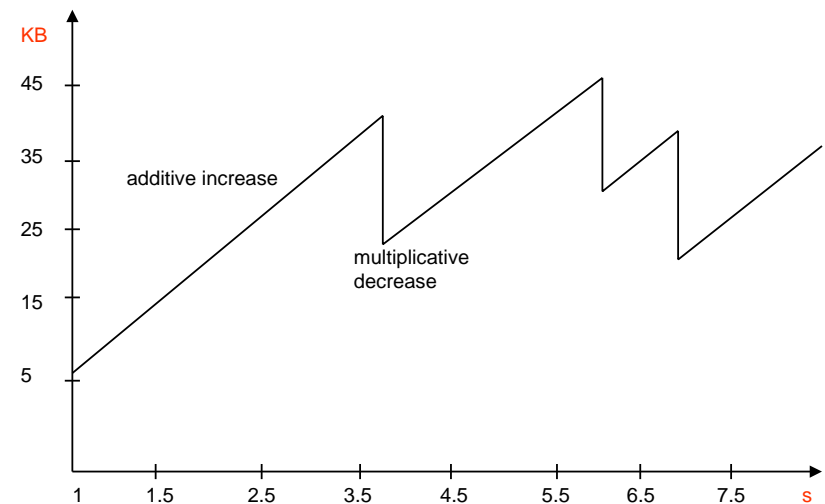
TCP Slow Start



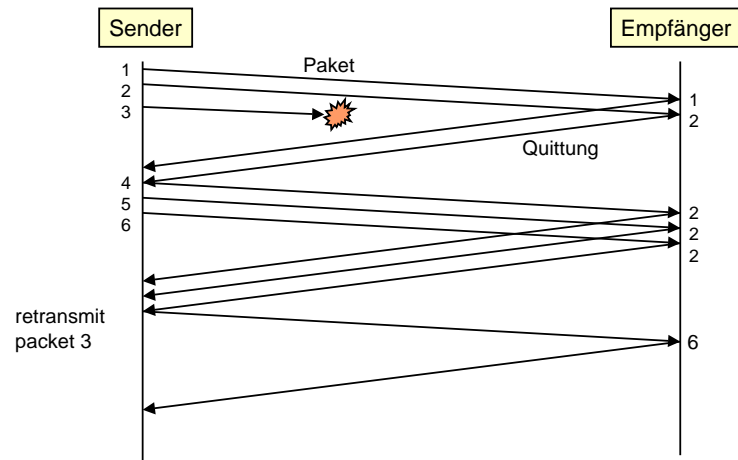
Additive Increase



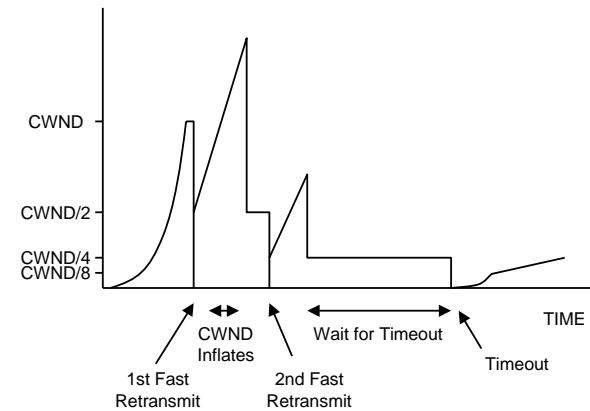
TCP Sawtooth Pattern



TCP Fast Retransmit

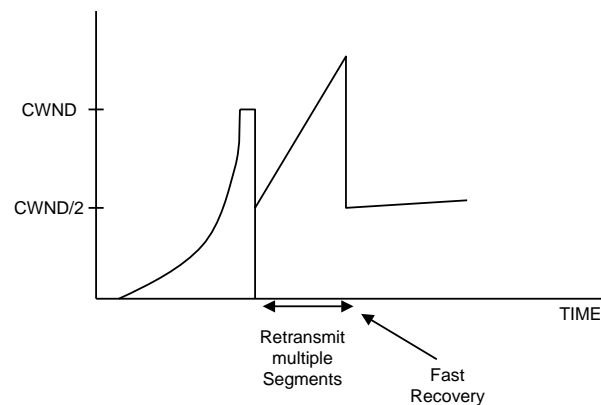


Fast Retransmit and Recovery



cwnd Congestion Window

Selective Acknowledgement



cwnd Congestion Window

TCP-Staukontrolle

Ziel

Vermeidung von Überlastsituationen im Netz

Anmerkung: Flusskontrolle vermeidet Überlast beim Empfänger

Aufgabe

- Keine Unterstützung durch das Netz
- Überlastsituationen können nur durch Beobachtung vermutet werden

Verfahren

- Bei ausbleibender Quittung wird Stausituation vermutet
- Daraufhin erfolgt eine Begrenzung der Datenmenge, die ein TCP-Sender senden darf
- Sender testet, wie viel Daten er senden kann, indem die Datenmenge langsam erhöht wird
- Hierzu: Einführung zweier neuer Variablen
 - Staukontrollfenster (CWnd)
 - Schwellenwert (SSTresh)

Explizite Staukontrolle

-Neu: Explizite Stauenachrichtigung für IP und TCP (RFC 3168)

TCP-Staukontrolle

- Es muss immer gelten
 $\text{LastByteSent} - \text{LastByteAcked} \leq \min \{ \text{CWnd}, \text{RcvWindow} \}$
- Schwellenwert SSTresh bestimmt wie Staukontrollfenster vergrößert wird

Ablauf

- Start: $\text{CWnd} = 1 \text{ MSS}$
- Solange $\text{CWnd} \leq \text{SSTresh}$ und Quittungen vor Timeout empfangen
- **Slow-Start**
 - Exponentielles Erhöhen des Staukontrollfensters
 - Verdopplung des Staukontrollfensters je Umlaufzeit
- Empfangene Quittung: $\text{CWnd} + = 1$
- $\text{CWnd} > \text{SSTresh}$ und Quittungen vor Timeout empfangen
- **Congestion Avoidance**
 - Lineares Erhöhen des Staukontrollfensters
 - Erhöhen des Staukontrollfensters um 1 je Umlaufzeit
- Empfangene Quittung: $\text{CWnd} + = 1 / \text{CWnd}$
- Timeout
 - $\text{SSTresh} := \text{CWnd} / 2$
 - $\text{CWnd} = 1 \text{ MSS}$

Beobachtungen

Congestion Avoidance

- **AIMD** (additive increase, multiplicative decrease)
 - Erhöhen des Fensters um 1 je Umlaufzeit
 - Erniedrigen des Fensters um Faktor 2 bei vermutetem Datenverlust (Quittung nicht rechtzeitig empfangen)
- Algorithmus zur Staukontrolle geht auf Van Jacobson zurück
- Modifikationen des ursprünglichen Algorithmus
- RFC 2581
- Staukontrollfenster wird initial auf 2 MSS gesetzt

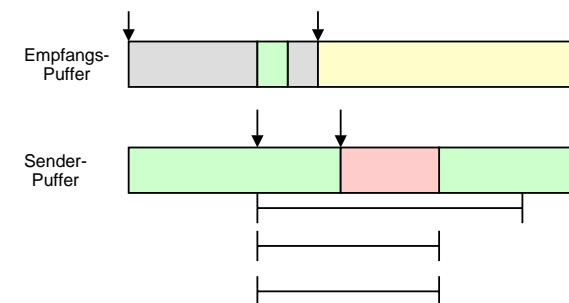
Staukontrolle

- Staukontrolle befasst sich mit Stausituationen in Zwischensystemen.
- Stau in Zwischensystemen führt zu Übertragungswiederholungen durch Transportprotokoll
 ⇒ Verstärkung der Stausituation

Ansätze in TCP

- Adaption des TCP-Sendefensters
 - Erhöhen des Sendefensters nach Erhalt einer Quittung
- exponentiell durch Verdopplung des Fensters am Anfang der Verbindung oder bei Rücksetzen nach längerer Stausituation (**slow start**)
- linear im Sättigungsbereich, d.h. Erhöhen des Fensters um 1 Paketgröße (**linear increase**)
 - Reduzieren der Fenstergröße nach Ausbleiben einer Quittung (Zeitüberwachung) auf die Hälfte (multiplicative decrease).
- Explicit Congestion Notification (RFC 2481)

Integration von Stau- und Flusskontrolle



- $\text{AdvertisedWindow} = \text{MaxReceiveBuffer} - (\text{LastByteReceived} - \text{LastByteRead})$
- $\text{MaxWindow} = \min (\text{CongestionWindow}, \text{AdvertisedWindow})$
- $\text{EffectiveWindow} = \text{MaxWindow} - (\text{LastByteSent} - \text{LastByteAcked})$

Tahoe, Reno und Vegas

Verschiedene Varianten der TCP-Staukontrolle

Tahoe

- der vorher beschriebene Algorithmus
- Problem
 - Sender muss bei Datenverlust evtl. lang auf den Timeout warten
 - Nicht jeder Datenverlust geht auf eine Stausituation zurück (z.B. aktives Warteschlangen-Management)

Reno

- umfasst **Fast-Retransmit**
 - Sendewiederholung wird angestoßen, falls 3 gleiche Quittungen empfangen wurden
- umfasst **Fast-Recovery**
 - Slow-Start-Phase wird nach Fast-Retransmit nicht verwendet
- ist in vielen Betriebssystemen implementiert

NewReno

- verbessert nochmals Fast-Recovery für Fälle, in denen mehrere Dateneinheiten nicht quittiert wurden.

Vegas

- Idee: Stausituationen erkennen bevor Datenverluste auftreten, dann lineare Reduzierung (anstelle der multiplikativen Erniedrigung bei Tahoe und Reno)
- Vermuten einer Stausituation basierend auf steigenden Umlaufzeiten
- heute wenig implementiert

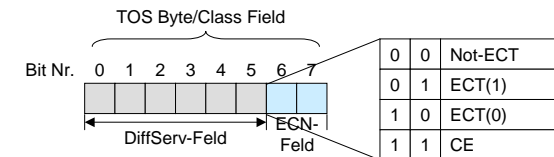
Staukontrolle: Explicit Congestion Notification

Problem: Netz ist „Black Box“ # Endsysteme schließen auf Stausituation nur indirekt über Paketverlust

Paketverlust als Stauanzeige für verzögerungssensitive Anwendungen (Remote Login) ungünstig (Timeout + RTT)

Lösung: IP-Erweiterung ECN [RFC 3168]

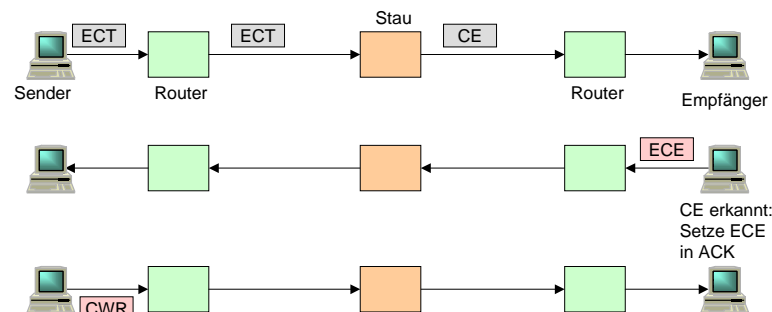
- Vermeiden von Paketverlusten durch explizite Stauanzeige des Netzes (signalisiert durch niederwertigste 2 Bit im ehemaligen IP-ToS-Feld)
 - Setzt Active Queue Management im Router voraus:
- Anzeige muss erfolgen, bevor Warteschlange wirklich voll ist
- Markierung des IP-Pakets (Congestion Experienced – CE) anstatt es zu verwerfen
 - ECN-Fähigkeit muss signalisiert werden, um Unfairness zu vermeiden: ECN-Capable Transport (ECT) Bits



Staukontrolle: ECN-Beispiel mit TCP

Erweiterung im TCP-Header: ECN-Echo-Flag (ECE) und Congestion-Window-Reduced-Flag (CWR)

Bei Verbindungsaufbau: ECN-Fähigkeit von TCP durch SYN+ECE+CWR signalisiert



ECE erkennt:

Halbiere Staufenster, reduziere Slow-Start-Schwellenwert
Setze CWR in der nächsten Dateneinheit

Adaptive Übertragungswiederholung

- Wiederholung von Segmenten, falls innerhalb eines bestimmten Zeitintervalls (Timeout) keine Quittung empfangen wird.

- Timeout wird abhängig von RTT gesetzt.

Original-Algorithmus

- $\text{SampleRTT} = \text{Zeit zwischen Senden eines Segments und Empfang der Quittung}$
- $\text{EstimatedRTT} = a * \text{EstimatedRTT} + (1 - a) * \text{SampleRTT}$
- $0.8 < a < 0.9$
- $\text{Timeout} = 2 * \text{EstimatedRTT}$
- Problem: Bezieht sich ACK bei Übertragungswiederholung auf erstes oder wiederholtes Segment?

