Digitale Kommunikation und Internetdienste 1

Wintersemester 2004/2005 – Teil 9

Belegnummer Vorlesung: 39 30 02

Übungen: 39 30 05

Jan E. Hennig

AG Rechnernetze und Verteilte Systeme (RVS)
Technische Fakultät
Universität Bielefeld

jhennig@rvs.uni-bielefeld.de

basierend auf den Arbeiten von Michael Blume, Heiko Holtkamp, Marcel Holtmann und I Made Wiryana

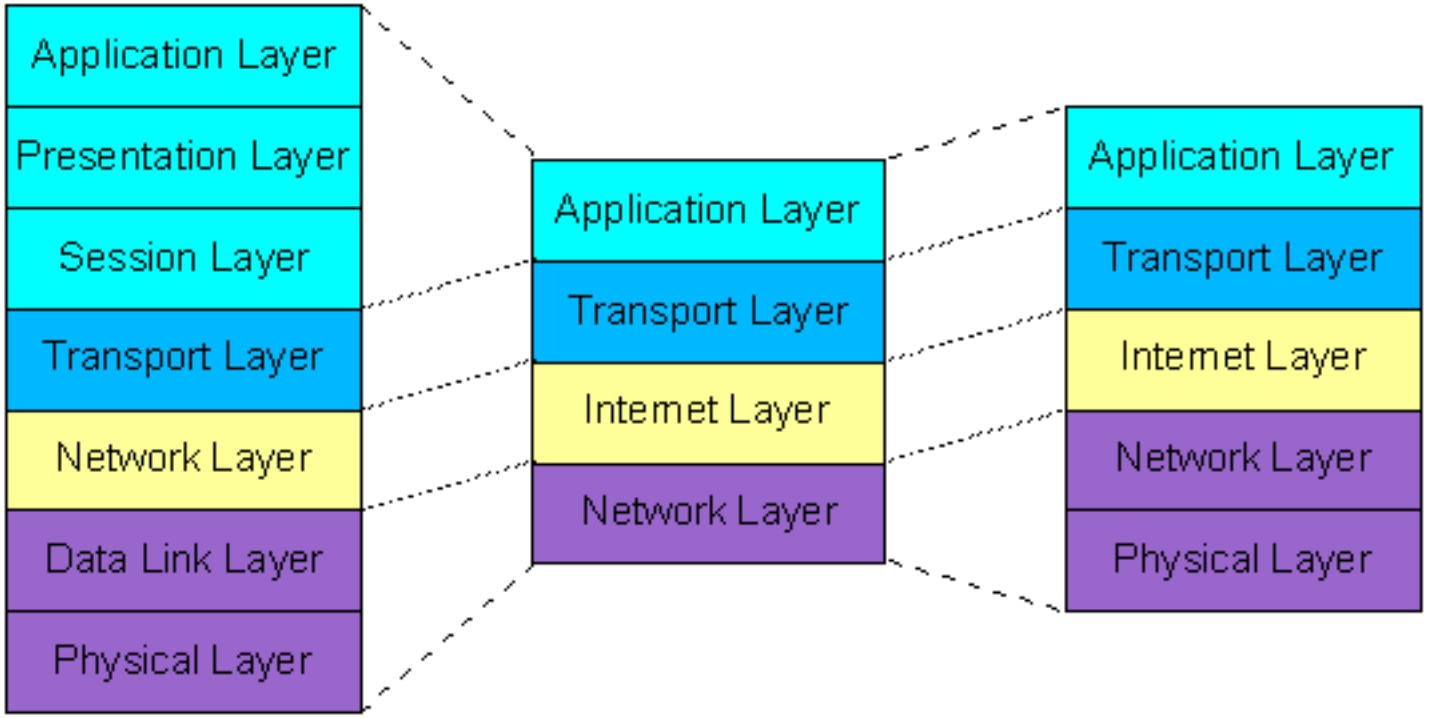
Erinnerung (1)

- Netzwerkschicht:
- physikalische Übertragung von Bits und Bytes
- grundlegende Fehlerkorrektur
- Kollisionsbehandlung oder -vermeidung
- verschiedene Techniken (Ethernet, Token Ring, ...)
- zueinander inkompatibel

Erinnerung (2)

- Internetschicht:
- Internetworking: Verbinden verschiedener Netzwerke
- Vereinheitlichung durch Abstraktion
- logische Adressierung und Routing
- Fragmentierung zu großer Pakete

Modellüberblick



Übersicht

- Transportschicht:
- Multiplexing/Demultiplexing
- UDP
- Zuverlässige Übertragung:
- Stop-and-Wait-Algorithmus
- ARPANET-Algorithmus
- Sliding-Window-Algorithmus
- Ausblick auf TCP

Kanäle

- bislang betrachtet: jeweils ein Kanal
- tatsächlich benötigt: mehrere Kanäle
- denn: mehr als ein Prozeß je Knoten
- oder ein Prozeß benötigt mehrere Verbindungen gleichzeitig
- jedoch nur 1 Medium

Multiplexing

- Methode: Multiplexing/Demultiplexing
- teilen eines Mediums
- a) pro Zeiteinheit
- b) pro Frequenz
- c) . . .
- benötigt Informationen zum Wiederzuordnen
- hier: Teilen von Zeit (pro Paket)

Prozeß-zu-Prozeß

- das einfachste Transportprotokoll erweitert den Host-zu-Host-Zustelldienst
- zu einem Prozeß-zu-Prozeß-Kommunikationsdienst
- viele Prozesse laufen gleichzeitig auf einem Host
- benötigt wird also ein (De)Multiplexing für Prozesse
- so daß sich mehrere Prozesse einen Netzwerkdienst teilen können

UDP (1)

- *User Datagram Protocol (UDP)*
- definiert in RFC 768
- einfachstes Transportprotokoll
- fügt keine weitere Funktionalität außer der Prozeßzuordnung ein
- daher: unzuverlässiges, verbindungsloses Protokoll (nach „best effort“)

UDP (2)

- Protokoll stellt keinerlei Mechanismen zur Verfügung, die sichern, daß die Daten auch tatsächlich beim Zielrechner ankommen
- sind die Daten aber beim Zielrechner angekommen, so sind sie auch korrekt
- UDP bietet gegenüber TCP den Vorteil eines geringen Protokoll-Overheads
- viele Anwendungen, bei denen nur eine geringe Anzahl von Daten übertragen wird, verwenden UDP als Transportprotokoll
- da unter Umständen der Aufwand zur Herstellung einer Verbindung und einer zuverlässigen Datenübermittlung größer ist
- als die wiederholte Übertragung der Daten
- z.B. Client/Server-Anwendungen, die auf der Grundlage einer Anfrage und einer Antwort laufen

UDP (3)

- UDP benutzt die Prozeßnummern eines Hosts nicht direkt
- dies würde voraussetzen, daß Prozeßnummern einheitlich sind
- jedoch unterscheiden sie sich von Betriebssystem zu Betriebssystem
- daher: Abstraktion und Übersetzung in logische Nummerierung
- genannt: *Port*nummern

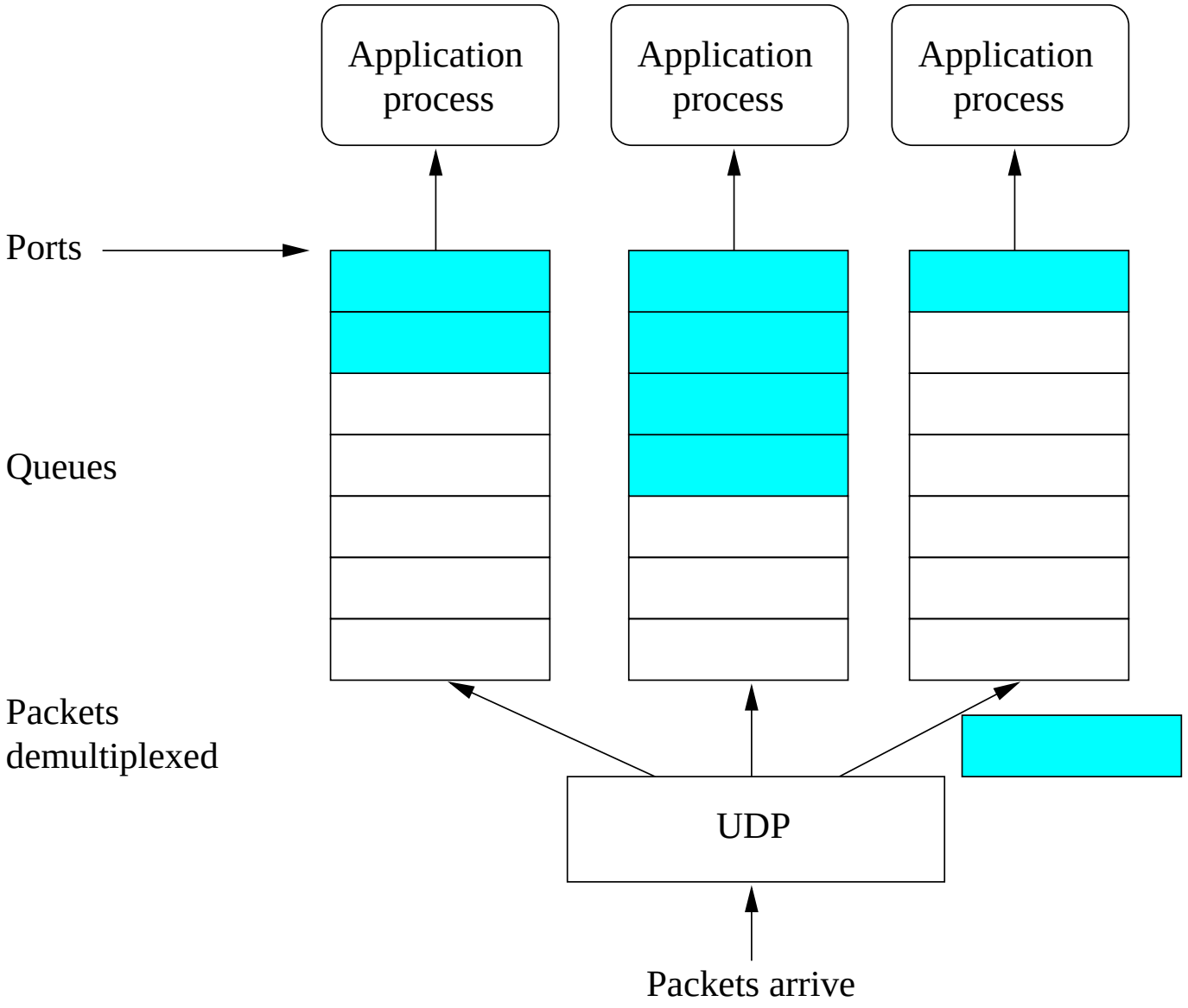
UDP (4)

- UDP verwendet 16 Bit für eine Portnummer
- somit je Host bis zu 65536 Ports benutzbar
- dies reicht nicht, um alle Prozesse im Internet zu identifizieren
- jedoch nur Identifikation je Host nötig
- Host selbst wird über Adresse der Internetschicht identifiziert

UDP (5)

- wie die Portzuordnung zu Prozessen geschieht, unterscheidet sich von Betriebssystem zu Betriebssystem
- typischerweise wird ein Port als Nachrichten-Warteschlange implementiert
- trifft eine Nachricht ein, so wird sie hinten an die Schlange angeführt
- ist kein Platz mehr vorhanden, so wird die Nachricht verworfen
- es gibt keinen Kontrollmechanismus, der dem Sender sagen könnte, er möge langsamer senden
- ist die Schlange leer, so wird der Prozeß blockiert, bis eine Nachricht eintrifft

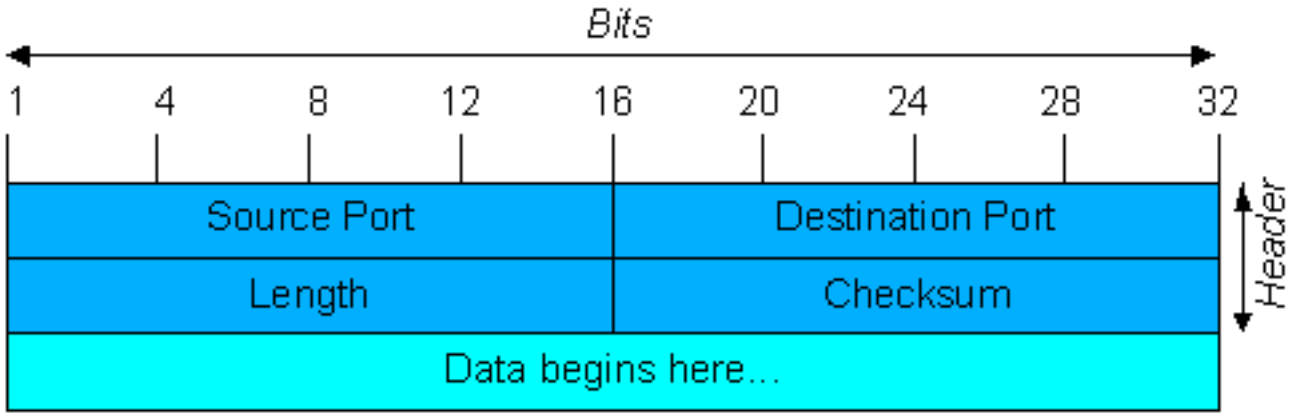
UDP (6)


Application process
Application process
Application process
Ports
Queues
Packets demultiplexed
UDP
Packets arrive

UDP (7)

- UDP-Header: 8 Byte
- Felder im UDP-Header:
- *Quellport (source port)*
- *Zielport (destination port)*
- *Prüfsumme (checksum)*, optional
- *Länge (length)*

UDP-Header (1)


Bits
1
4
8
12
16
20
24
28
32
Source Port
Destination Port
Length
Checksum
Data begins here...
Header

UDP-Header (2)

- Quell- und Zielport:
- bezeichnen logische Prozessnummer relativ zum Quell- bzw. Zielhost
- Quellport ist optional (Wert=0 bei Nichtverwenden)
- Länge:
- bezeichnet Anzahl Bytes des gesamten Datengramms
- also Header + Nutzdaten

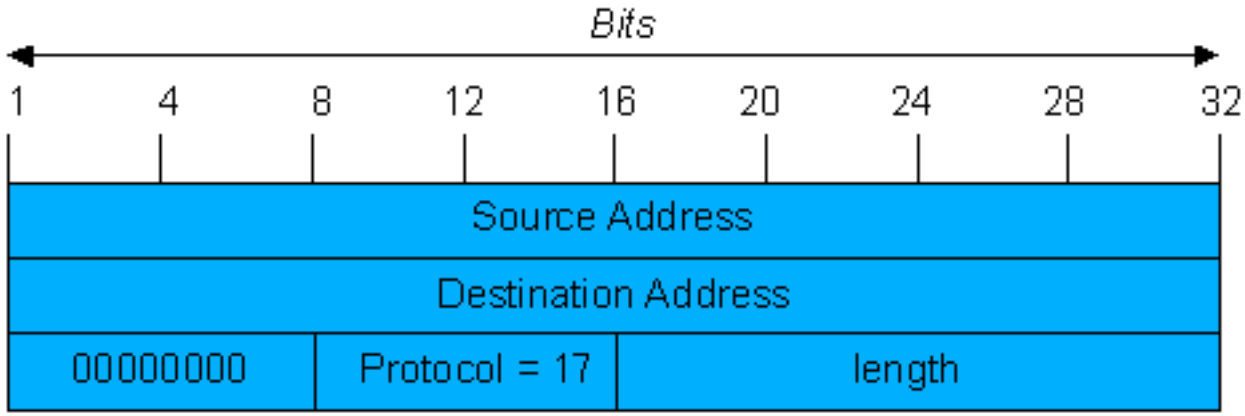
UDP-Header (3)

- Prüfsumme:
- Feld ist optional (Wert=0 bei Nichtverwenden)
- Algorithmus wie beim Internet Protocol
- alle 16-Bit Wörter werden im 1er-Komplement addiert und die Summe ermittelt
- die Berechnung wird durchgeführt über
  - den UDP-Header
  - die Daten
  - und den *Pseudo-Header*

UDP-Pseudo-Header (1)

- der Pseudo-Header enthält:
- die 32-bit großen IP-Adressen der Quell- und Zielmaschine
- sowie die Protokollnummer (für UDP 17)
- und die Länge des UDP-Pakets

UDP-Pseudo-Header (2)



UDP-Pseudo-Header (3)

- die Einbeziehung der IP-Adressfelder des Pseudo-Headers in die Prüfsummenberechnung hilft, durch IP falsch zugeteilte Pakete zu erkennen
- jedoch führt bereits IP eine solche Berechnung durch
- (IPv4, nicht jedoch IPv6)
- die Verwendung von IP-Adressen auf der Transportebene stellt eine Verletzung der Protokollhierarchie dar
- entsprechend muß beim Umstellen auf IPv6 auch UDP geändert werden

Portnummern (1)

- Problem: Woher bekommt man die Portnummern für die Kommunikation?
- bei einer Antwort auf eine Anfrage kann ehem. Quellport benutzt werden
- aber wie beginnt die Kommunikation?
- Lösung: man weist Anwendungen bestimmte Portnummern fest zu
- für bekannte wichtige Anwendungen ist ein Nummernbereich reserviert

Portnummern (2)

- Festlegung sogenannter *well known ports*
- RFC 1700 und `www.iana.org`
- z.B. Nameserverdienst auf Port 53/udp
- Ports sind protokollspezifisch
- UDP-Ports unterhalb von 1024 sind festgelegt
- über 1024 liegt Bereich zur freien Verwendung

Auszug aus /etc/services


```
echo            7/udp
discard         9/udp           sink null
daytime         13/udp
msp             18/udp                          # message send protocol
chargen         19/udp          ttytst source
fsp             21/udp          fspd
ssh             22/udp                          # SSH Remote Login Protocol
time            37/udp          timserver
rlp             39/udp          resource        # resource location
re-mail-ck      50/udp                          # Remote Mail Checking Protocol
domain          53/udp          nameserver
bootps          67/udp
bootpc          68/udp
tftp            69/udp
gopher          70/udp
www             80/udp                          # HyperText Transfer Protocol
kerberos        88/udp          kerberos5 krb5 kerberos-sec     # Kerberos v5
csnet-ns        105/udp         cso-ns
#3com-tsmux     106/udp         poppassd
rtelnet         107/udp
pop2            109/udp         pop-2
pop3            110/udp         pop-3
sunrpc          111/udp         portmapper      # RPC 4.0 portmapper UDP
ntp             123/udp                         # Network Time Protocol
pwdgen          129/udp                         # PWDGEN service
netbios-ns      137/udp
netbios-dgm     138/udp
netbios-ssn     139/udp
imap2           143/udp         imap
```

Zuverlässige Übertragung

- bisher vorgestellt: unzuverlässige Übertragung
- Pakete können verloren gehen (z.B. verworfen werden wegen Übertragungsfehlern)
- für zuverlässige Übertragung müssen solche Pakete erneut gesendet werden
- TCP sorgt für zuverlässige Übertragung

Methoden

- es gibt verschiedene Methoden, um zuverlässige Übertragung sicherzustellen
- gemeinsam ist das Verwenden von:
- *Bestätigungen (acknowledgements, kurz: ACK)*
- *Zeitüberschreitungen (timeouts)*

ACKs

- Bestätigungen informieren den Sender, daß ein Paket erfolgreich angekommen ist
- falls gerade ein Paket in Gegenrichtung gesendet werden soll
- kann eine Bestätigung gleich huckepack mitgesendet werden
- sonst ist ein gesondertes Paket nötig

Timeouts

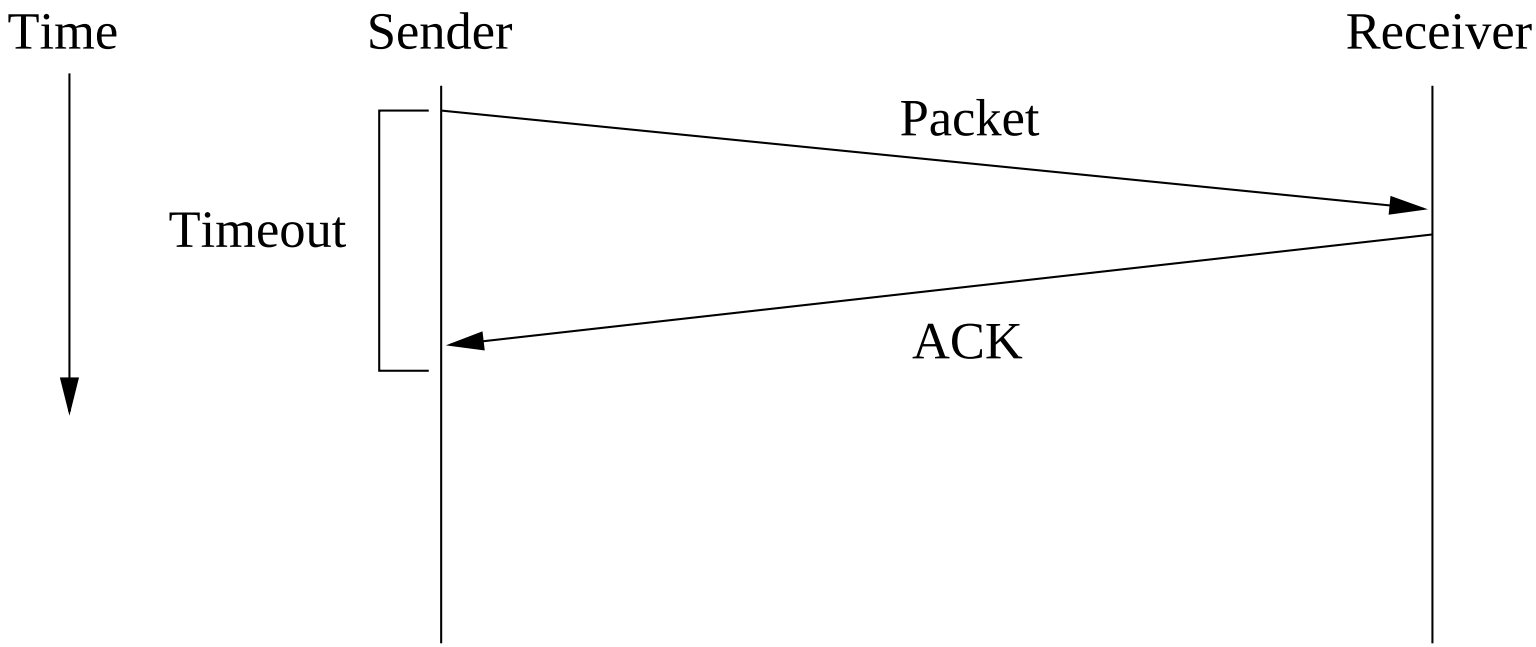
- erhält der Sender keine Bestätigung innerhalb eines festgelegten Zeitraums
- so sendet er das Paket erneut
- somit ergibt sich ein *automatic repeat request (ARQ)* durch Zeitablauf
- das Warten selbst wird auch *timeout* genannt

Stop-and-Wait (1)

- *stop-and-wait*-Algorithmus ist einfachstes ARQ-Schema:
- nachdem ein Paket gesendet wurde, wartet der Sender auf ein ACK
- dann erst schickt er das folgende Paket
- erhält er kein ACK innerhalb eines festgelegten Zeitraums
- so schickt der Sender das Paket erneut

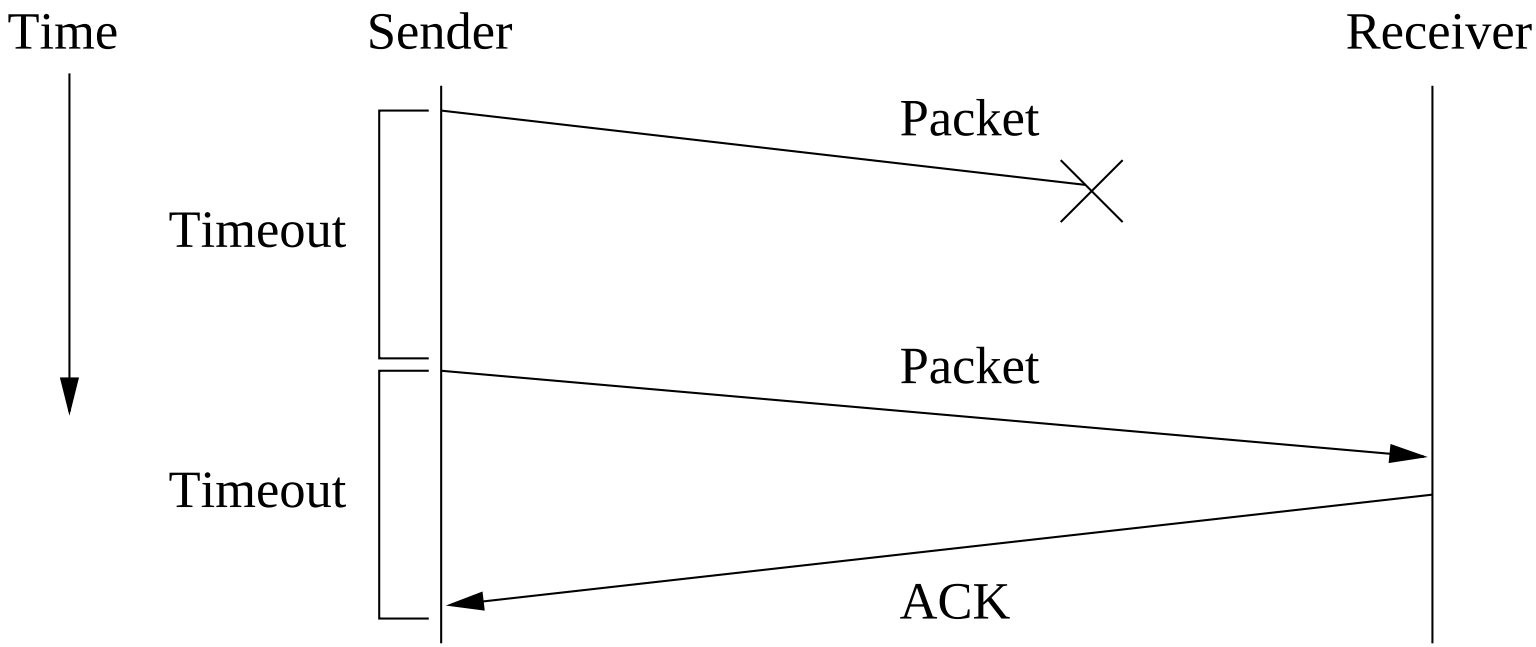
Stop-and-Wait (2)

(a) ACK wird vor Ablauf des Timeouts empfangen



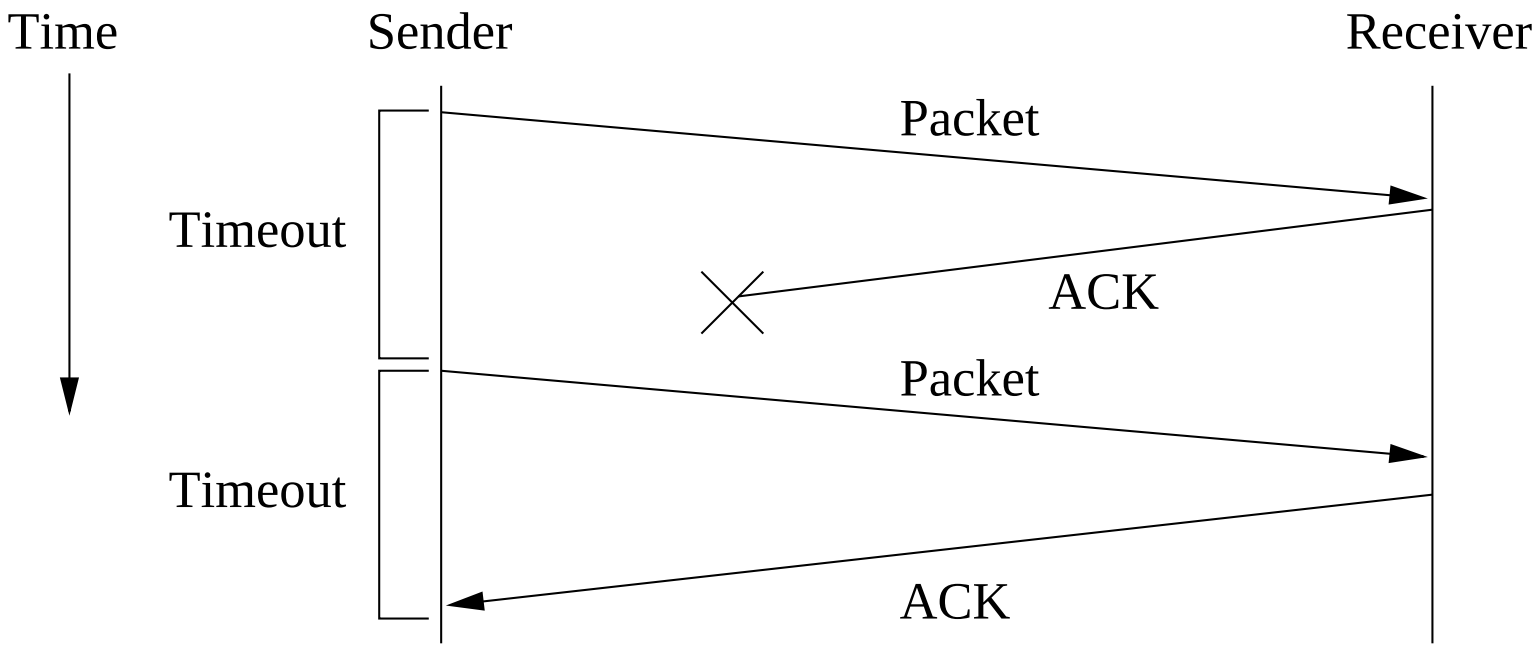
Stop-and-Wait (3)

(b) das erste Paket geht verloren



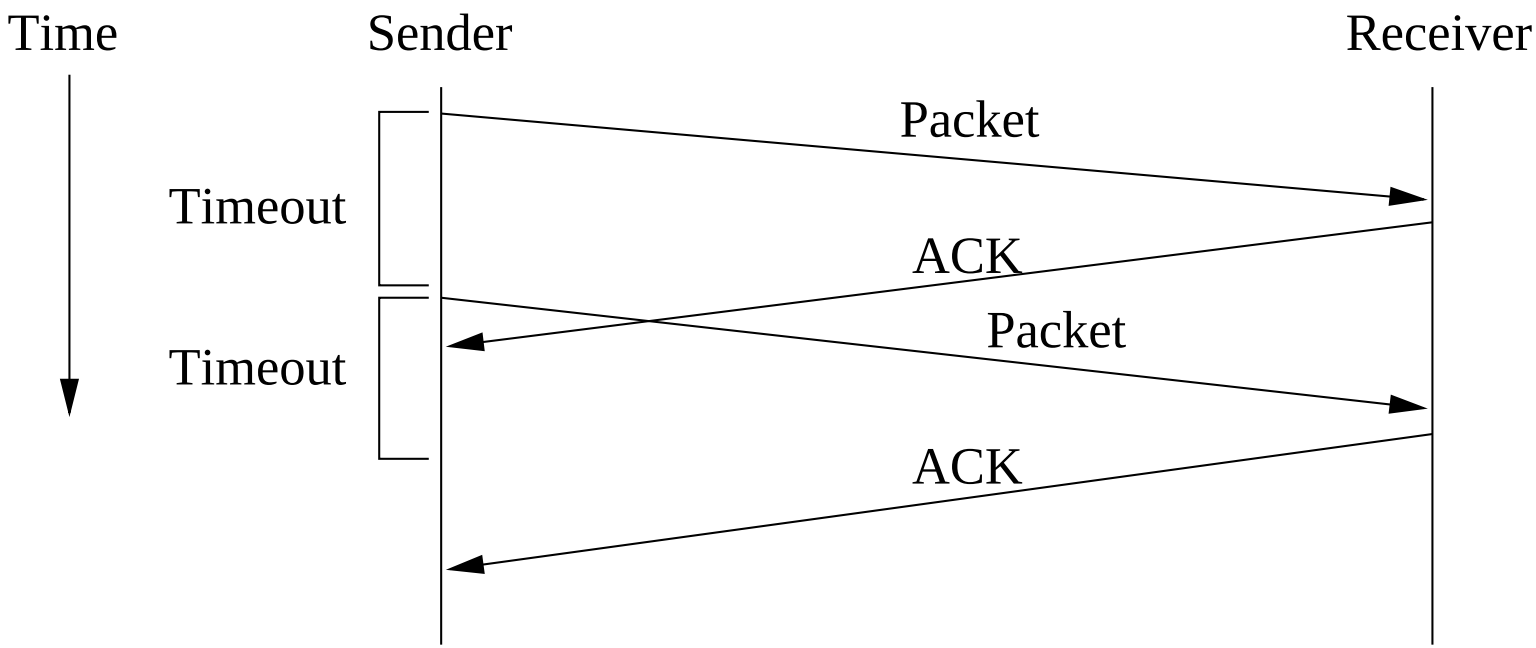
Stop-and-Wait (4)

(c) das ACK-Paket geht verloren



Stop-and-Wait (5)

(d) der Timeout ist zu kurz



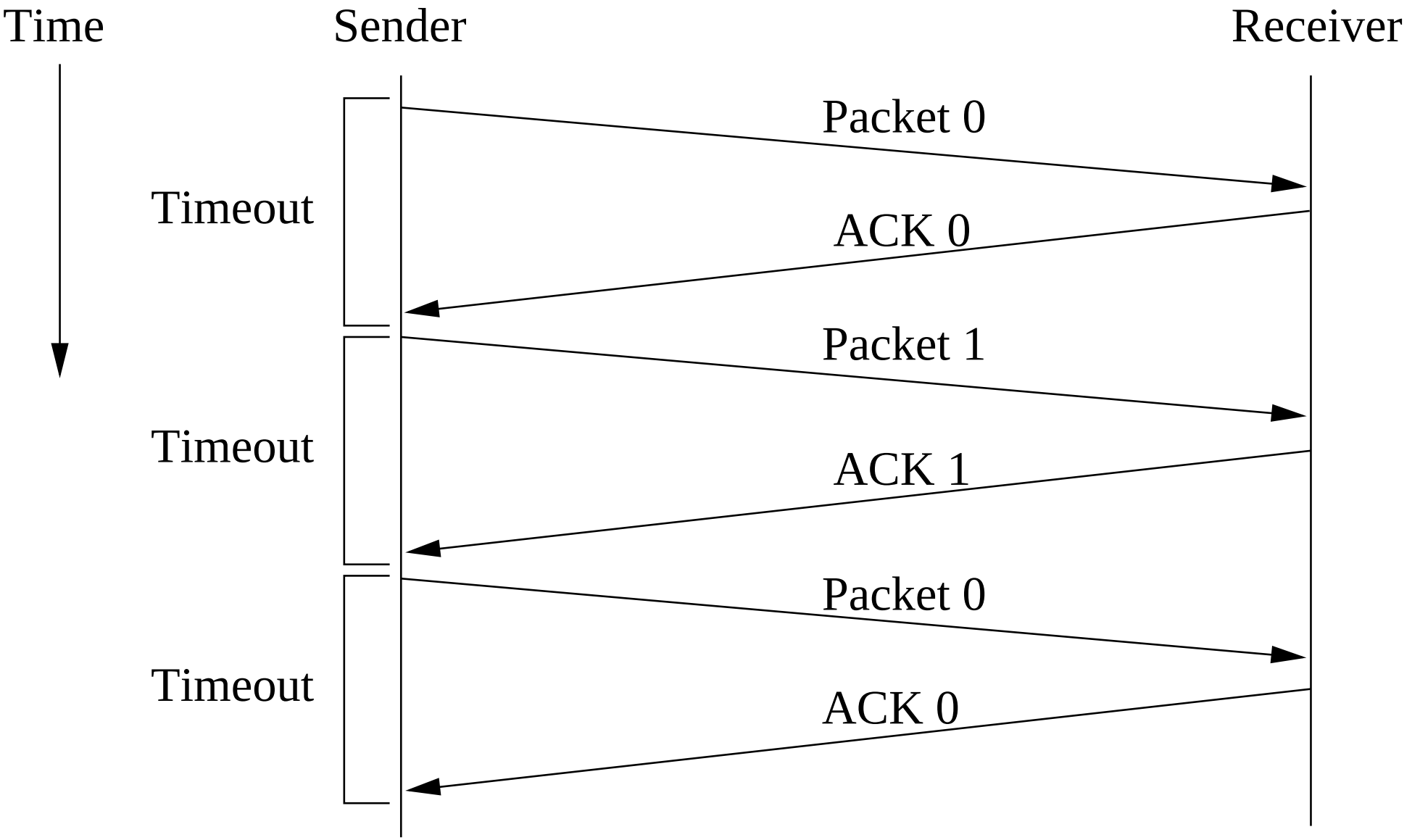
Stop-and-Wait (6)

- Problem bei Fällen (c) und (d):
- Empfänger hat Paket empfangen, jedoch Sender sendet es erneut
- Empfänger muß unterscheiden können, ob ein Paket das nächste Paket in der Reihe ist
- oder ob es eine Wiederholung ist
- sonst könnte er Duplikate nicht voneinander unterscheiden

Stop-and-Wait (7)

- Lösung:
- Einbinden einer Sequenznummer ins Paket
- diese wird in der Bestätigung wiederholt
- minimal: 1-Bit-Sequenznummer

Stop-and-Wait (8)



Stop-and-Wait (9)

- Unschönheit beim Stop-and-Wait-Algorithmus:
- der Sender kann immer nur ein Frame senden
- und muß das ACK abwarten
- → die Kapazität des Netzwerks wird nicht genutzt

Beispielrechnung

- 1,5 Mbps-Verbindung mit 45ms RTT
- delay $\times$ bandwidth $=$ 67,5Kb, ca. 8KB
- bei Paketgröße von 1KB und Stop-and-Wait (1 Paket pro RTT):
- BitsProPaket / ZeitProPaket = $1024 \times 8/0,045 = 182$Kbps
- also nur etwa $\frac{1}{8}$ der möglichen Kapazität

Verbesserung

- beim ARPANET konnte die Kapazität ausgenutzt werden
- ARPANET-Algorithmus: *Concurrent Logical Channels*
- mehrere logische Kanäle werden in eine einzige Punkt-zu-Punkt-Verbindung multiplext
- für jeden Kanal wird einzeln der Stop-and-Wait-Algorithmus durchgeführt
- zwischen den Paketen verschiedener Kanäle besteht keine Verbindung

ARPANET-Algorithmus (1)

- auf jedem Kanal kann ein Paket zum Versenden bereitstehen
- solange Daten bereitstehen, kann die Kapazität ausgenutzt werden
- ARPANET-Sender verwendeten 3 Bit, um den Kanalzustand darzustellen:
  - busy
  - next sequence number to send
  - expected ACK sequence number
- gesendet wird immer vom Kanal mit niedrigster Nummer, der nicht *busy* ist.

ARPANET-Algorithmus (2)

- in der ARPANET-Praxis:
- 8 logische Kanäle über erdgebundene Verbindungen (3 Bit-Kanalnummer)
- Paket enthält dafür Feld mit 3-Bit-Kanalnummer und 1-Bit-Sequenznummer
- 16 logische über satellitengebundene Verbindungen (4-Bit-Kanalnummer)

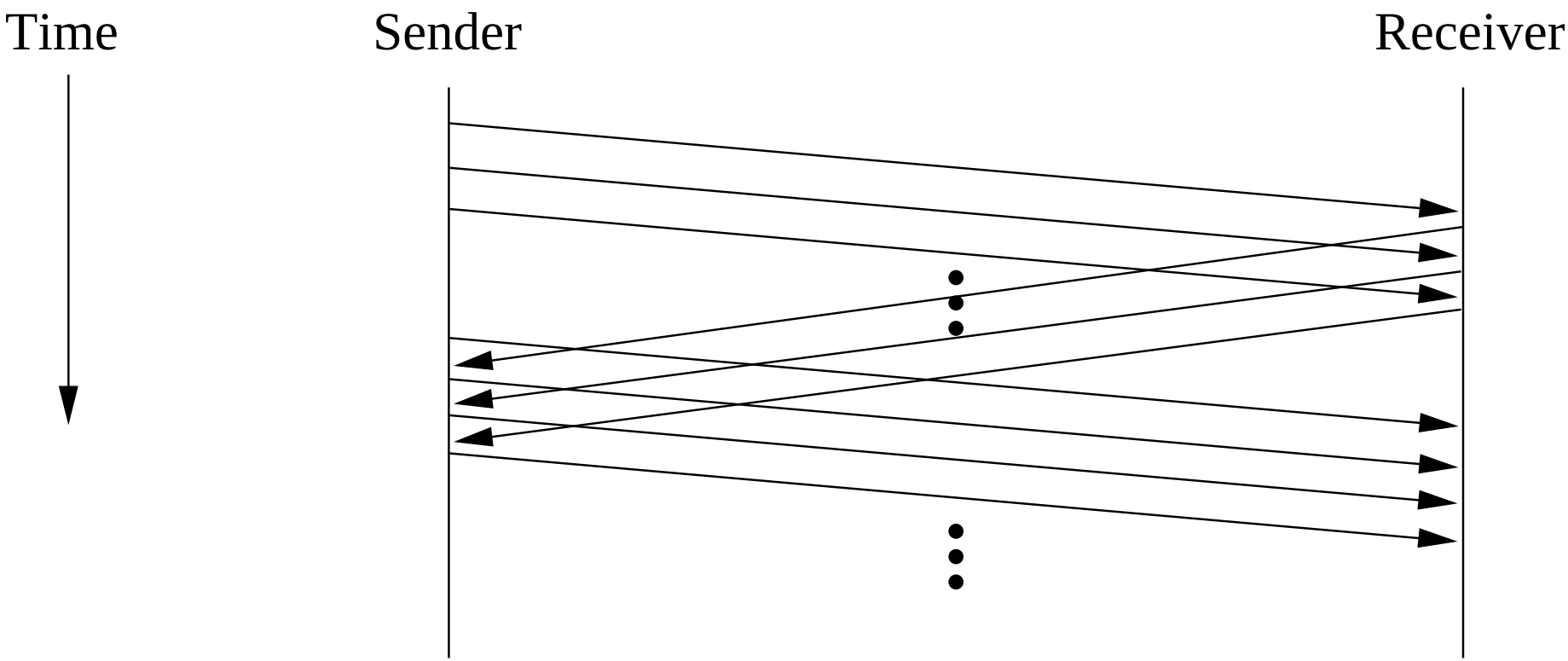
Alternative

- Problem damit: Reihenfolge nicht vorhersagbar
- feste Kanalvorgaben: keine dynamischen Anpassungen möglich
- Alternative:
- Erlaubnis mehr als nur ein unbestätigtes Paket zu senden
- Anzahl möglicher Pakete: *Sendefenster*
- → *Sliding-Window*-Algorithmus

Sliding Window (1)

- Beispielrechnung: 8 KB möglich bei 1 KB-Paketen
- $\frac{1}{8}$ der Kapazität benutzt mit Stop-and-Wait
- sinnvolle Erlaubnis:
- Sender darf 8 Pakete senden und 9. Paket vorhalten
- 9. Paket wird versendet, wenn Sender ACK für Paket 1 erhält

Sliding Window (2)



Sliding Window (3)

- jedes Paket erhält eine eindeutige Sequenznummer
- Sender verwaltet drei Variablen:
  - *send window size (SWS)*
  - *last acknowledgement received (LAR)*
  - *last frame sent (LFS)*
- Empfänger verwaltet ebenfalls drei Variablen.
  - *receive window size (RWS)*
  - *largest acceptable frame (LAF)*
  - *last frame received (LFR)*
- außerdem beim Empfänger: *SeqNumToAck* (größte fortlaufende Sequenznummer, die noch nicht bestätigt wurde)

Sliding Window (4)

- Sender muß eine Bedingung einhalten:
- $LFS - LAR \leq SWS$
- Empfänger muß andere Bedingung einhalten:
- $LAF - LFR \leq RWS$

Sliding Window (5)

Sliding Window (6)

- Sender führt Buch über jedes gesendete Paket (eigener Timeout)
- LAR wird erhöht, wenn ACK-Paket eintrifft
- alle Pakete im SWS müssen zwischengespeichert werden
- denn sie müssen evtl. erneut verschickt werden

Sliding Window (7)

- trifft ein Paket mit Sequenznummer s ein, so prüft der Empfänger
- $s \leq LFR \quad \vee \quad s > LAF$
- falls diese Bedingung zutrifft wird das Paket verworfen
- sonst: Paket prüfen und ACK senden
- Bestätigung erfolgt mit *SeqNumToAck*[1]
- selbst wenn Pakete mit höherer Sequenznummer empfangen wurden

[1] größte fortlaufende Sequenznummer, die noch bestätigt werden muß

Sliding Window (8)

- Bestätigung ist *kummulativ*
- anschließend anpassen:
- LFR = SeqNumToAck
- LAF = LFR + RWS

Beispiel

- z.B. LFR=5, RWS=4, LAF=9
- treffen Pakete 7 und 8 ein, wird kein ACK gesendet
- denn Paket 6 steht noch aus
- 7 und 8 sind außer der Reihe angekommen
- trifft Paket 6 nun ein, so wird Paket 8 bestätigt
- dann wird angepaßt: LFR := 8; LAF := 12

Effizienz

- treffen alle ACKs ein, so wird die Leitungskapazität ausgenutzt
- wichtig dafür ist, daß SWS/RWS passend zur Kapazität gewählt werden
- kommt es zum Timeout, so sinkt der Durchsatz
- Sender kann SWS nicht verschieben, solange Paket 6 nicht bestätigt wurde
- je länger es dauert dies festzustellen, desto größer ist dieses Problem

Effizienzsteigerungen

- man könnte beim Empfang von Paket 7 ein *negative acknowledgement (NAK)* senden
- ausreichend aber ist bereits allein der Timeout-Mechanismus
- NAKs erhöhen die Komplexität beim Sender
- man könnte auch ACKs auf Paket 5 senden, wenn Paket 7 und 8 ankommen
- in einigen Fällen kann der Sender daraus auf Paketverlust schließen
- erreicht werden kann so eine gewisse Früherkennung von Paketverlust

Selektive ACKs

- man könnte auch nicht-kummulative ACKs verwenden
- sondern selektive ACKs
- dann würden Pakete 7 und 8 bestätigt
- je mehr Information der Sender erhält, desto eher kann er die Leitungskapazität ausnutzen
- jedoch müssen in Gegenrichtung somit auch mehr Pakete verschickt werden
- und: Komplexität insgesamt steigt

Sliding Window

- Sliding-Window-Algorithmus wird von TCP benutzt
- es gilt aber noch weitere Probleme zu lösen
- z.B. welche Größen soll man für SWS und RWS wählen?
- z.B. Sequenznummern sind nicht unendlich, sondern endlich

Aussicht

Themenübersicht für die kommende Vorlesung:

- Transportschicht:
- Probleme und Lösungen für Sliding-Window-Algorithmus
- TCP

Ende Teil 9. Danke für die Aufmerksamkeit.