Datenkommunikation

Transportschicht
TCP (1)

Wintersemester 2011/2012

Einordnung

1	Grundlagen von Rechnernetzen, Teil 1
2	Grundlagen von Rechnernetzen, Teil 2
3	Transportzugriff
4	Transportschicht, Grundlagen
5	Transportschicht, TCP (1)
6	Transportschicht, TCP (2) und UDP
7	Vermittlungsschicht, Grundlagen
8	Vermittlungsschicht, Internet
9	Vermittlungsschicht, Routing
10	Vermittlungsschicht, Steuerprotokolle und IPv6
11	Anwendungsschicht, Fallstudien
12	Mobile IP und TCP

Überblick

1. **Einordnung und Aufgaben des Protokolls**
2. Der TCP-Header
3. Verbindungsauf- und abbau
4. Datenübertragung

Zusammenfassung zur Transportschicht

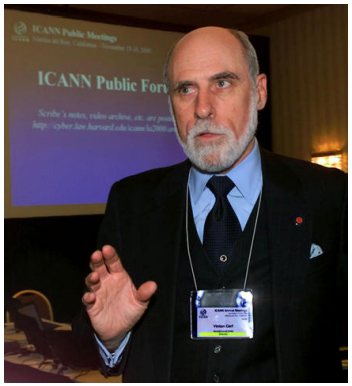
- Logischer Ende-zu-Ende-Transport
- Verbindungslos versus verbindungsorientiert
- Verbindungsmanagement und Adressierung
  - Verbindungsaufbau
  - Verbindungsabbau
- Zuverlässiger Datentransfer
  - Quittierungsverfahren
  - Übertragungswiederholung
- Flusskontrolle
- Staukontrolle

Überblick

- **Einordnung und Aufgaben des Protokolls**
- Der TCP-Header
- Verbindungsauf- und abbau
- Datenübertragung

Robert E. Kahn

Cinton G. Cerf

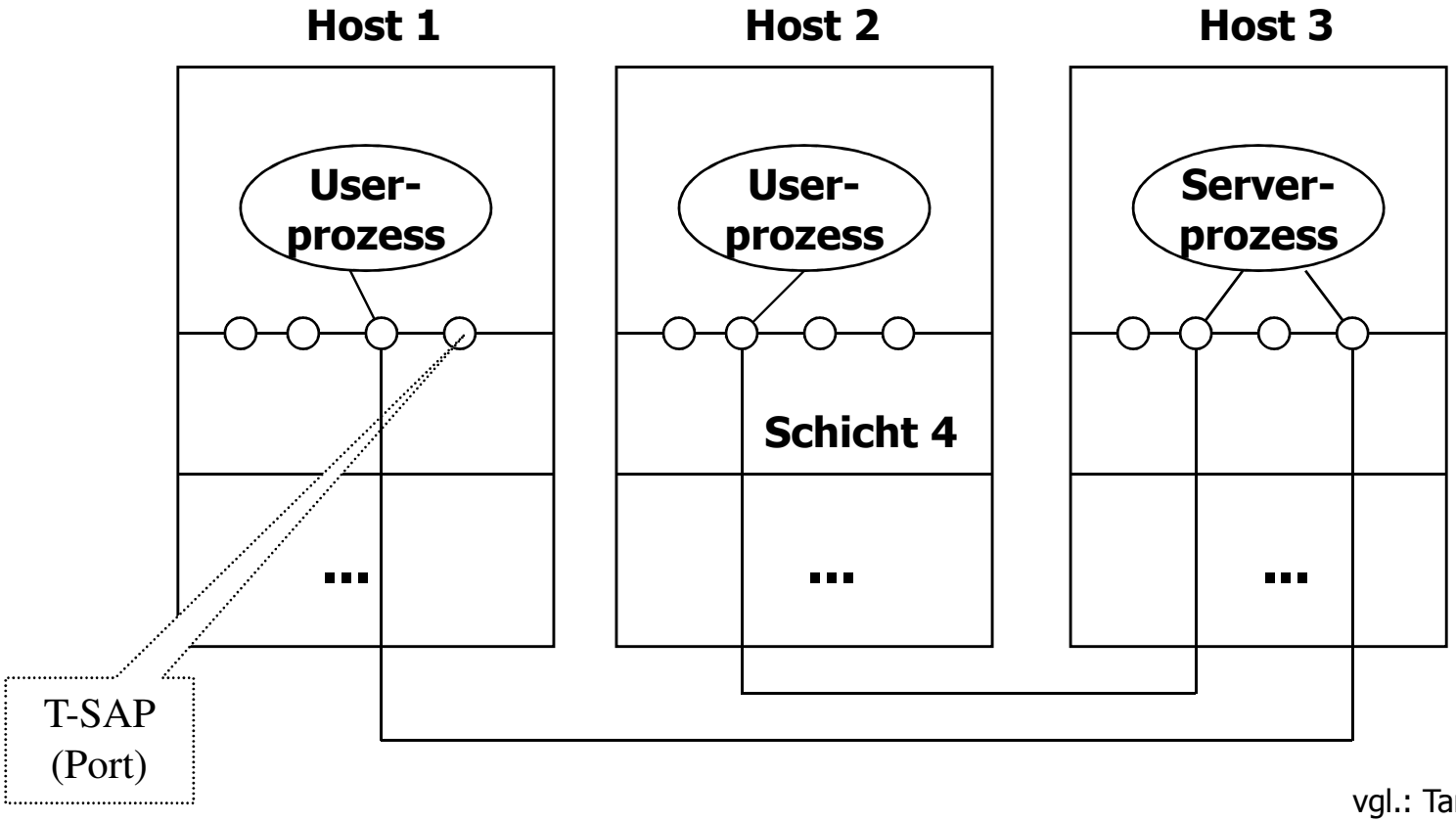


Geschichte:

- TCP wurde von Robert E. Kahn und Vinton G. Cerf ab 1972 entwickelt (mehrere Jahre)
- Erste Standardisierung von TCP 1981 im RFC 793

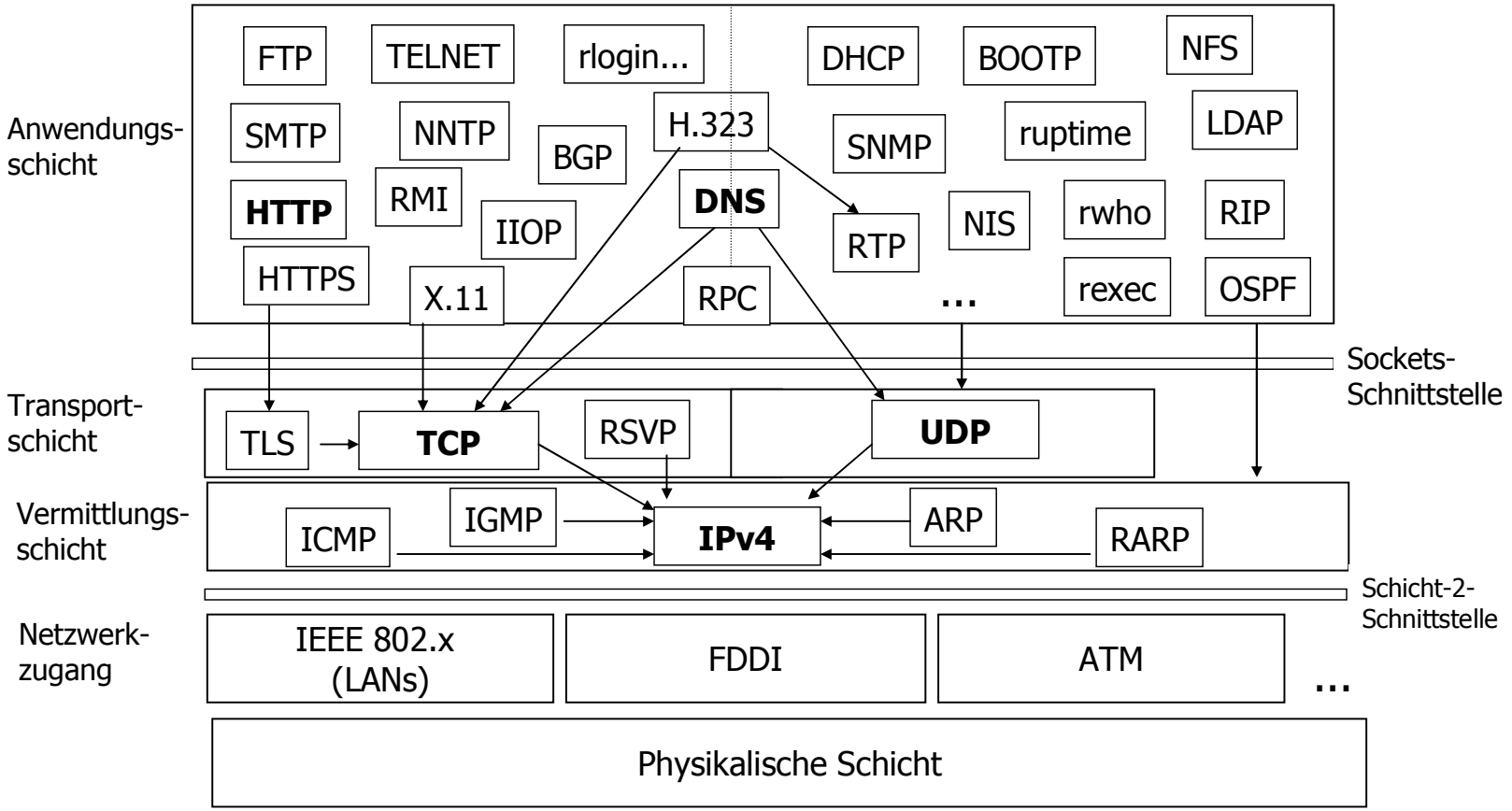
Einordnung und Aufgaben

- TCP ermöglicht eine **Ende-zu-Ende Beziehung** zwischen kommunizierenden Anwendungsinstanzen

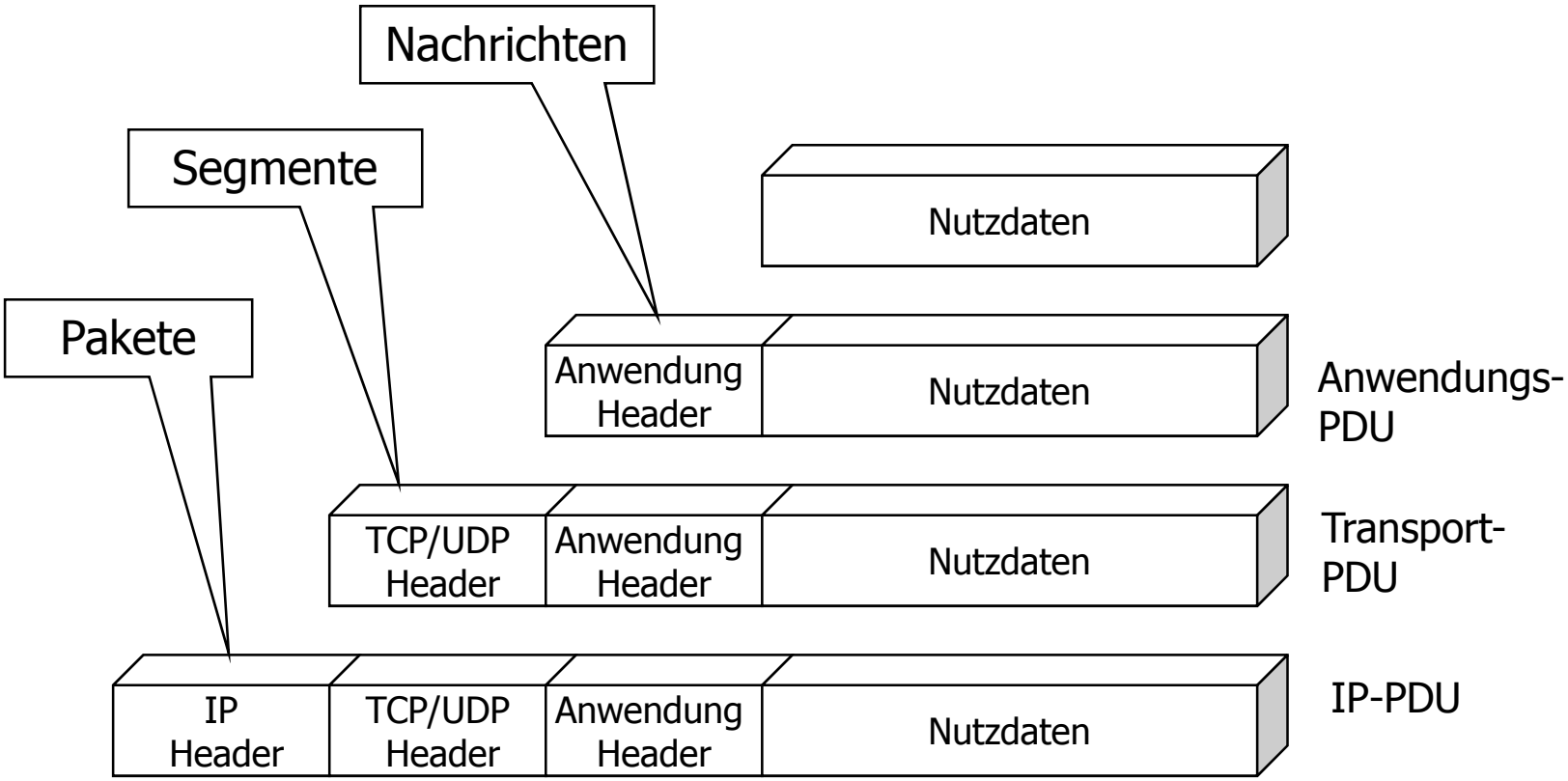


vgl.: Tanenbaum

Erinnerung: TCP/IP-Protokollfamilie



Erinnerung:
TCP/IP-Referenzmodell, Protokollkapselung



PDU = Protocol Data Unit

Einordnung und Aufgaben

- TCP ist weit **verbreitet**
  - Hersteller haben TCP/IP als **Industrienorm** akzeptiert
  - Genormt in RFC 793, aktualisiert in RFC 1122, 1349,…
- TCP ist **offen**, nicht proprietär
  - also nicht an einen Hersteller gebunden
- TCP ist **skalierbar**
  - Ein bestehendes Netz auf TCP/IP-Basis kann relativ einfach um weitere Rechner erweitert werden; die neuen Rechner benötigen nur IP-Adressen
- TCP/IP ist die **Grundlage des Internets**
  - TCP/IP ist Bestandteil von UNIX
  - Das Internet basiert auf TCP/IP

Einordnung und Aufgaben

- TCP ermöglicht die Kommunikation
  - über **vollduplex-fähige**, **bidirektionale**,
  - und **virtuelle Verbindungen** zwischen Anwendungen
- **Stream-orientierte** Kommunikation im Unterschied zur blockorientierten Übertragung (siehe OSI TP4)
- TCP bietet eine **zuverlässige Ende-zu-Ende-Verbindung** zwischen Stationen
- TCP verwendet IP als Grundlage des Pakettransports

Aufgaben im Detail

- Schaffung einer gesicherten Ende-zu-Ende-Verbindung auf Basis von IP
- Reihenfolgegarantie
- Garantierte Auslieferung
- Staukontrolle
- Flusskontrolle (Vermeidung von Überschwemmungen langsamer Empfänger)
- Multiplexing und Demultiplexing der IP-Verbindung
- Fragmentierung und Defragmentierung der Nachrichten (Segmente!)

Einordnung und Aufgaben

- **Maßnahmen zur Sicherung** der Übertragung:
  - Drei-Wege-Handshake für Verbindungsmanagement
  - Prüfsumme
  - Quittierung (ACK-PDU): Positiv-kumulativ und implizites NAK
  - Zeitüberwachung für jedes Segment (Timer) und Nachrichtenwiederholung
  - Go-Back-N für die Nachrichtenwiederholung
  - Sequenznummern = Folgenummern für die Reihenfolgeüberwachung
  - Sliding Windows Prinzip zur Flusskontrolle
  - Slow-Start-Verfahren

Dienste, Ports und Adressierung

- Anwendungsprozess kommuniziert über eine Adresse, die als **Socket** bezeichnet wird
  - Tupel der Form (IP-Adresse, TCP-Portnummer)
- **Well-known Ports**
  - 16-Bit-Integer
  - Es gibt hier eine Reihe von sog. well-known Ports für reservierte Services (Portnr. <= 1024)
  - Jeder Dienst hat eine eigene Portnummer (siehe Datei /etc/services unter Unix)
- Ein Anwendungsprozess kann auch mehrere Verbindungen unterhalten
- Client-Server-Prinzip leicht realisierbar:
  - Server wartet an einem Port auf Connect-Requests

Dienste, Ports und Adressierung

- Eine Verbindung wird durch ein Paar von Endpunkten identifiziert (**Socket Pair**)
- Dadurch ist es möglich, dass **ein TCP-Port** auf einem Host für **viele Verbindungen** genutzt werden kann
- Beispiel:
  - HTTP-Port 80 wird für viele Verbindungen eines HTTP-Servers verwendet. TCP-Verbindungen aus Sicht des HTTP-Servers sind:

    ((195.214.80.76, **80**) (196.210.80.10,6000))

    ((195.214.80.76, **80**) (197.200.80.11,6001))

    ...

    wobei 195.214.80.76 die IP-Adresse des Servers ist.

Einschub:

- ICANN-Vorgabe für die Ports:
  - http://www.iana.org/assignments/port-numbers
  - Well Known Ports
  - Registered Ports
  - Dynamic and/or Private Ports

  - Datei anschauen!

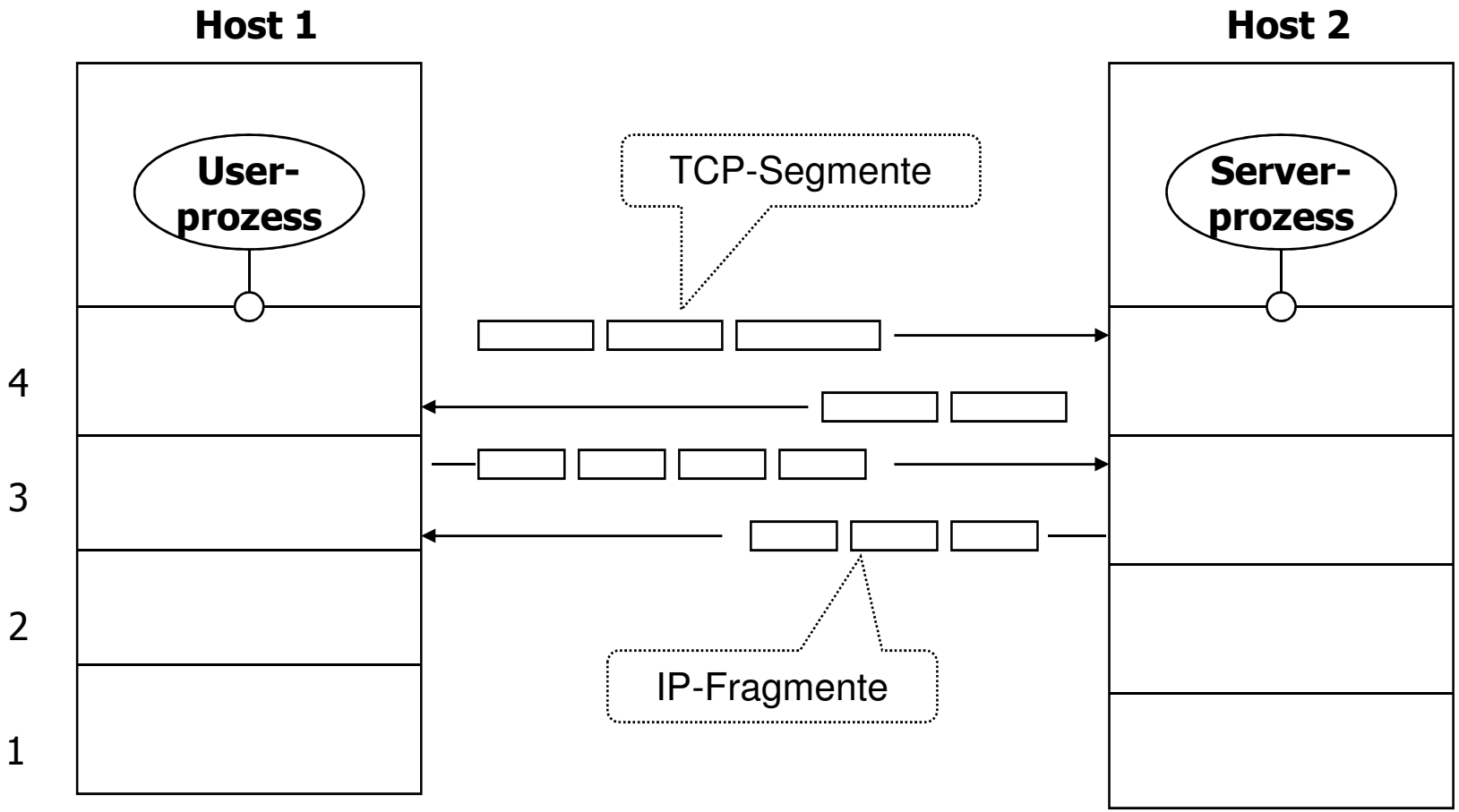
Beispiele für Dienste und Ports

TCP-Portnummer	Protokoll, Service
23	Telnet – Remote Login
20,21	ftp – File Transfer Protocol
25	SMTP – Simple Mail Transfer Protocol
80	HTTP

Überblick

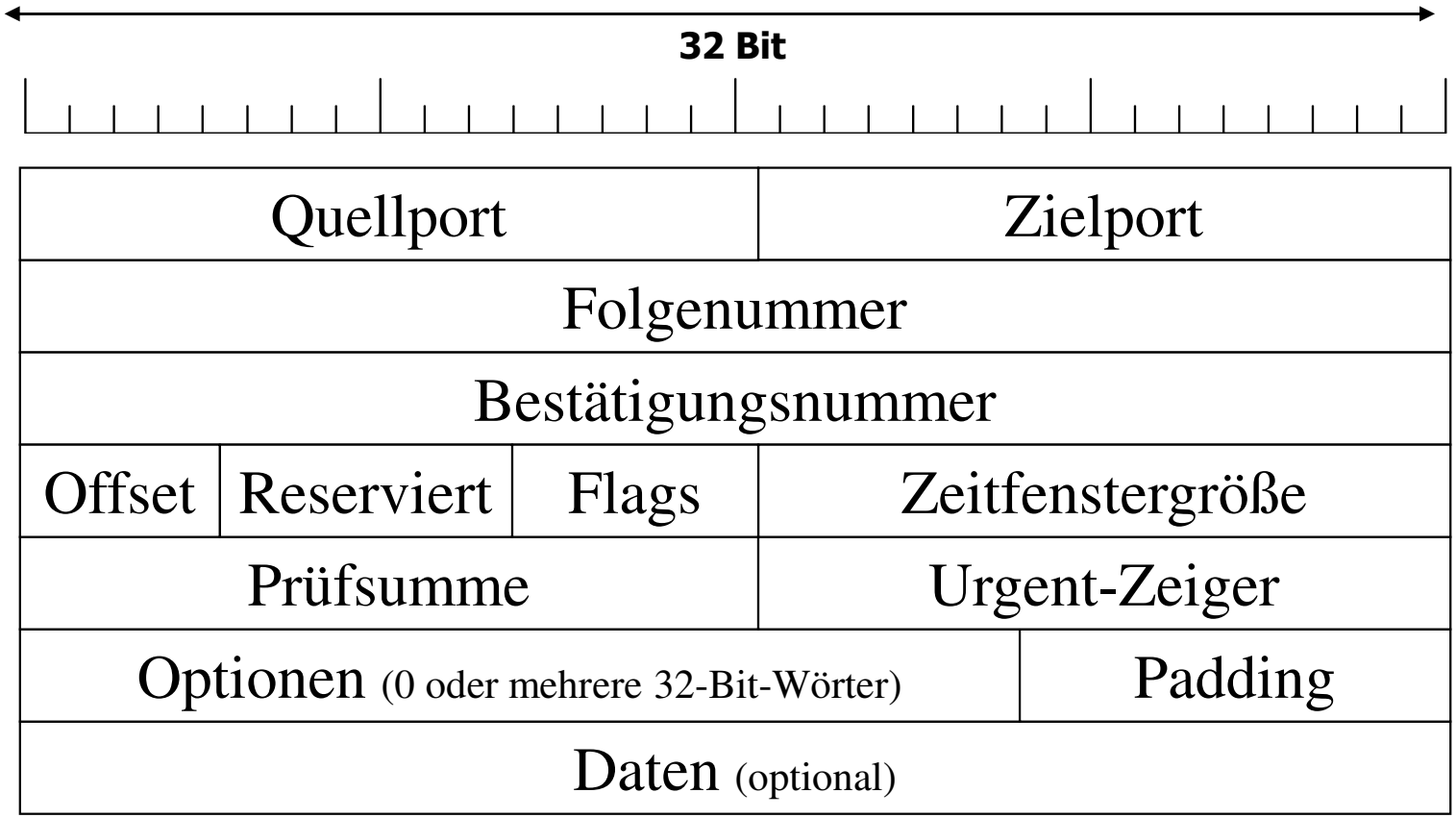
1. Einordnung und Aufgaben des Protokolls
2. **Der TCP-Header**
3. Verbindungsauf- und abbau
4. Datenübertragung

TCP-Segmente



- Byte-Strom wird in TCP-Segmente gepackt
- TCP-Segmentierung != IP-Fragmentierung

TCP-Header (PCI, Protocol Control Information)



TCP-Segmente

- TCP sieht den Datenstrom als eine Sequenz von Octets (Bytes) und unterteilt diese zur Übertragung in **Segmente**
- Ein Segment besteht aus einem mind. 20 Bytes langen Header
- Es bleiben max. 65.535 - 20 (TCP-Header), also **65.515 Bytes** für die Daten in einem Segment
- **Maximum Segment Size** (MSS) kann optional ausgehandelt werden
  - MSS-Option
  - Nicht verwechseln mit Window-Größe für Flusskontrolle, verlängerbar über WSOpt-Option, bis 1 GB

TCP-Header

- **Quell- und Zielport**
  - Portnummer des Anwendungsprogramms des Senders und des Empfängers
- **Folgenummer**
  - Nächstes Byte innerhalb des TCP-Streams (mod 2^{32})
- **Bestätigungsnummer**
  - Gibt das als nächstes erwartete Byte im TCP-Strom an und bestätigt damit den Empfang der vorhergehenden Bytes
- **Offset**
  - Gibt die Länge des TCP-Headers in 32-Bit-Worten an
- **Reserviert**
  - Hat noch keine Verwendung

TCP-Header

- **Flags**: Hier handelt es sich um Kontroll-Bits mit unterschiedlicher Bedeutung:

Flag	Bedeutung
URG	Urgent-Zeiger-Feld ist gefüllt
ACK	Bestätigung (z.B. bei Verbindungsaufbau genutzt), d.h. die ACK-Nummer hat einen gültigen Wert
PSH	Zeigt Push-Daten an, Daten dürfen beim Empfänger nicht zwischengespeichert werden, sondern sind sofort an den Empfängerprozess weiter zu leiten
RST	Dient zum - Rücksetzen der Verbindung (sinnvoll z.B. bei Absturz eines Hosts) - Abweisen eines Verbindungsaufbauwunsches - Abweisen eines ungültigen Segments
SYN	Wird genutzt beim Verbindungsaufbau
FIN	Wird genutzt beim Verbindungsabbau

TCP-Header

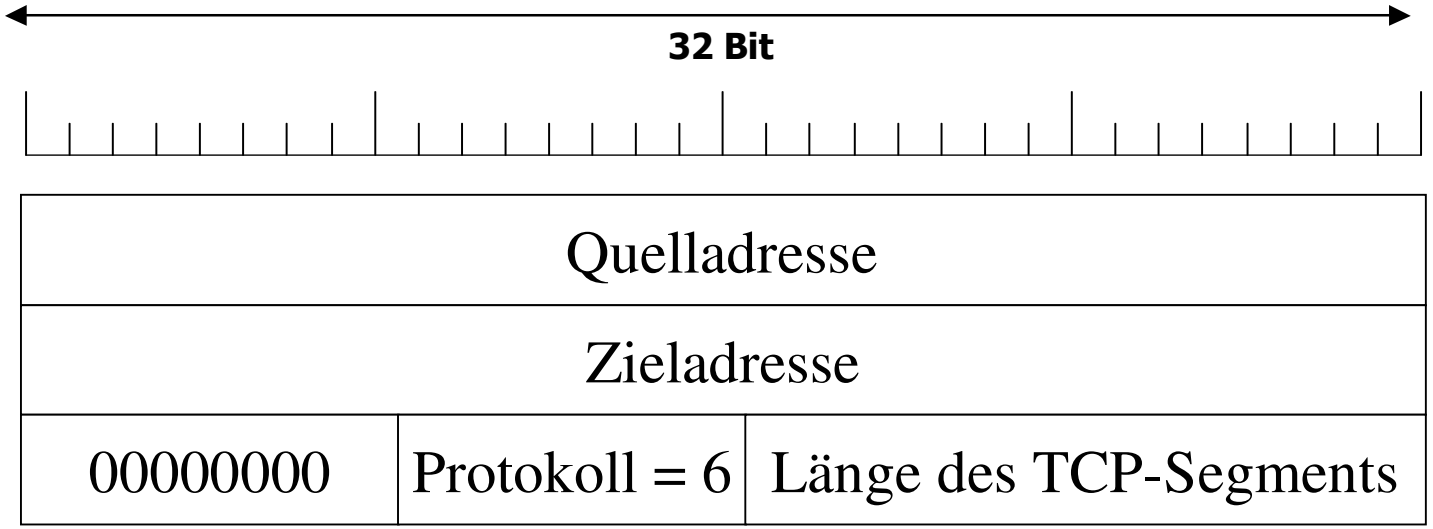
- **Zeitfenstergröße**
  - Erlaubt dem Empfänger, mit ACK dem Sender den vorhandenen Pufferplatz in Byte zum Empfang der Daten mitzuteilen
- **Urgent-Zeiger**
  - beschreibt die Position (Byteversatz von der aktuellen Folgenummer ab) an der dringliche Daten vorgefunden
  - Diese Daten werden vorrangig behandelt
  - Selten genutzt!

TCP-Pseudoheader

- **Prüfsumme – Berechnung**
  - Verifiziert das TCP-Segment (Header + Daten) inkl. eines Pseudoheaders auf Basis eines einfachen Prüfsummenalgorithmus:
    - Prüfsumme im Header auf Null setzen
    - Nutzdaten ggf. auf gerade Byteanzahl mit einem Nullbyte ergänzen
    - Summe über alle 16-Bit-Wörter der ganzen Nachricht inkl. TCP-Header und sog. Pseudo-Header bilden
    - Danach Bildung des Einer-Kompliments aus der Summe ergibt die Prüfsumme

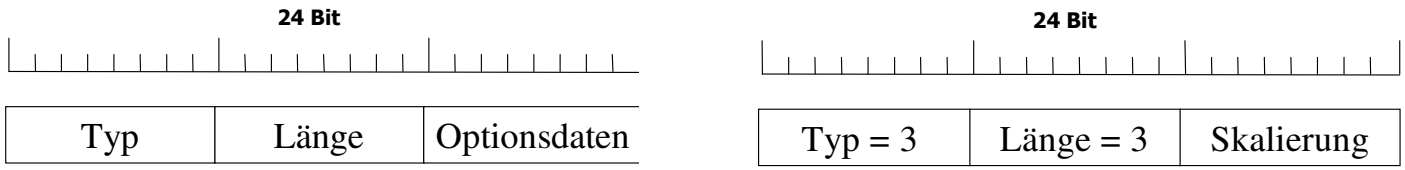
TCP-Pseudoheader

- **Pseudoheader:**
  - Wird vor der Berechnung der Prüfsumme an das TCP-Segment gehängt, aber nicht mit übertragen
  - Aufbau:

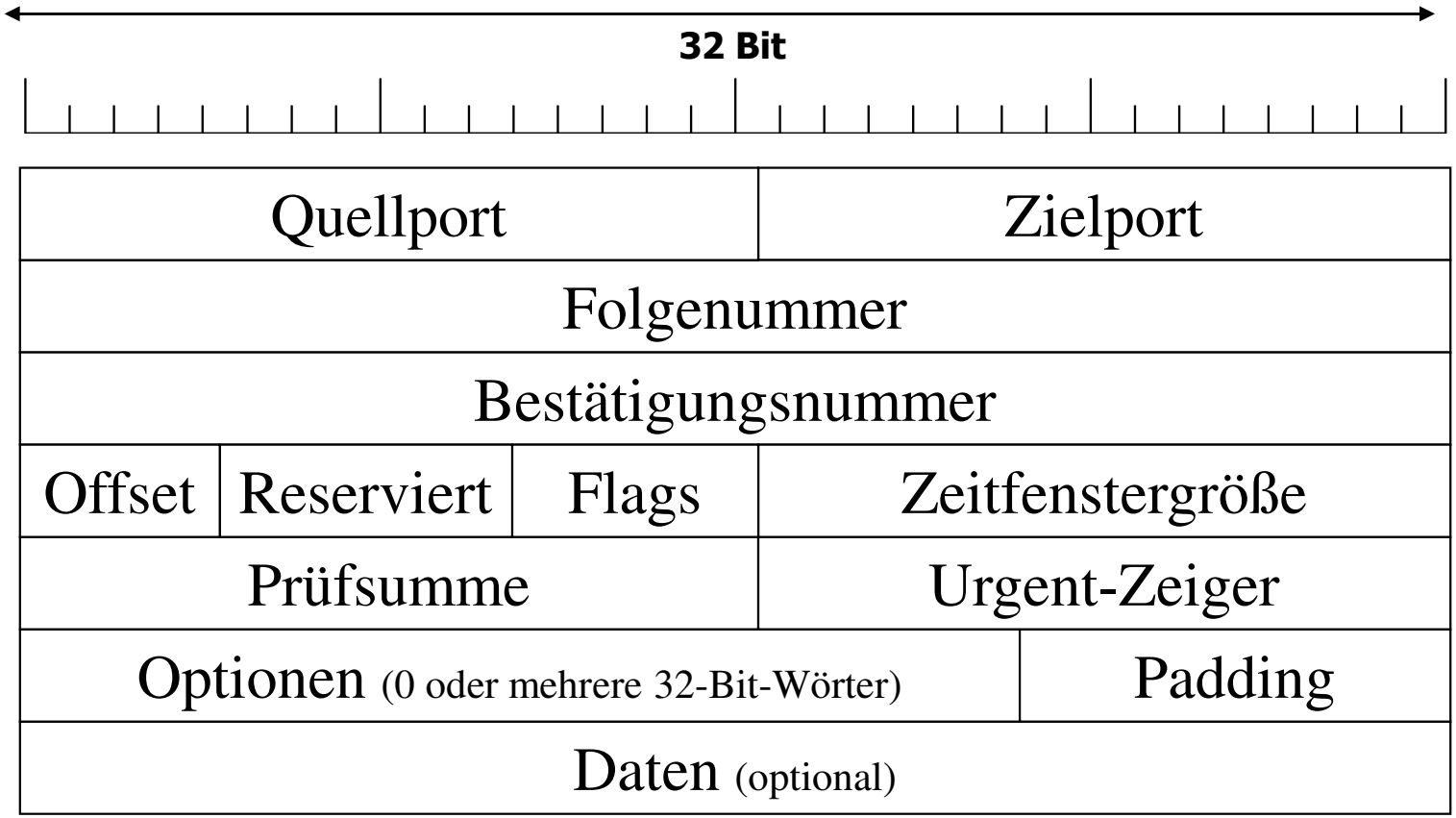


TCP-Header

- **Optionen**
  - Wurden früher relativ selten verwendet
  - Bestehen entweder aus einem einzelnen Byte (Optionsnummer) oder haben eine variable Form
  - Optionen sind z.B.:
    - MSS: Maximum Segment Size der Verbindung einstellen
    - *SACKOK*: Selektive Wiederholungen anstelle von *go back n* einstellen
    - *WSOPT (Windows-Scale-Option):* Maximale Fenstergröße verlängern (2^{30} Byte max. möglich)



TCP-Header (PCI, Protocol Control Information)



Überblick

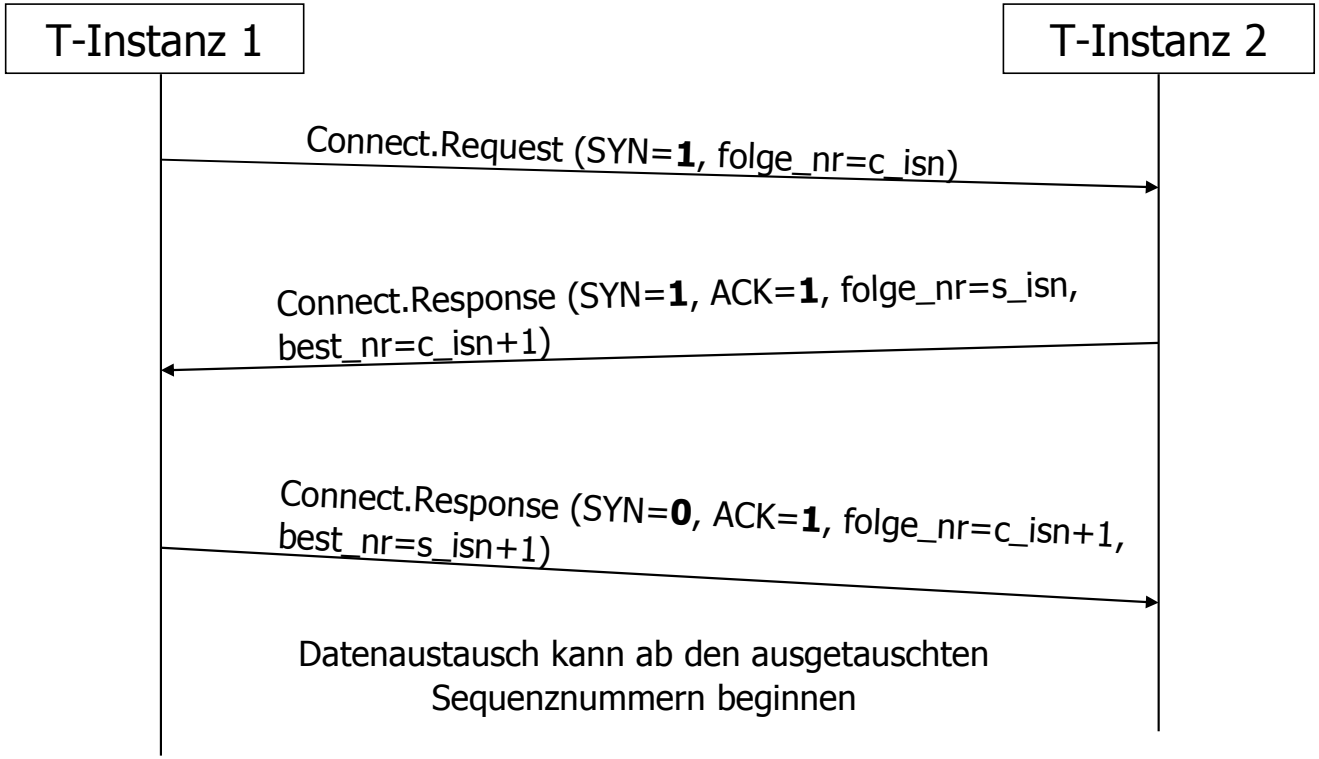
1. Einordnung und Aufgaben des Protokolls
2. Der TCP-Header
3. **Verbindungsauf- und abbau**
4. Datenübertragung

Verbindungsaufbau: Parameter aushandeln und Drei-Wege-Handshake

- Beim Verbindungsaufbau werden die **MSS** und die **Sequenznummern** ausgehandelt und ggf. weitere Einstellungen (Optionen) ausgehandelt
- Verwendung des Drei-Wege-Handshake-Mechanismus
  - Initiale Sequenznummern werden berechnet und ausgetauscht
- Kollision beim Verbindungsaufbau ist möglich:
  - Zwei Hosts versuchen gleichzeitig eine Verbindung mit gleichen Parametern zueinander aufzubauen
  - Es wird nur eine TCP-Verbindung aufgebaut

Verbindungsaufbau: Protokoll

- Normaler Ablauf



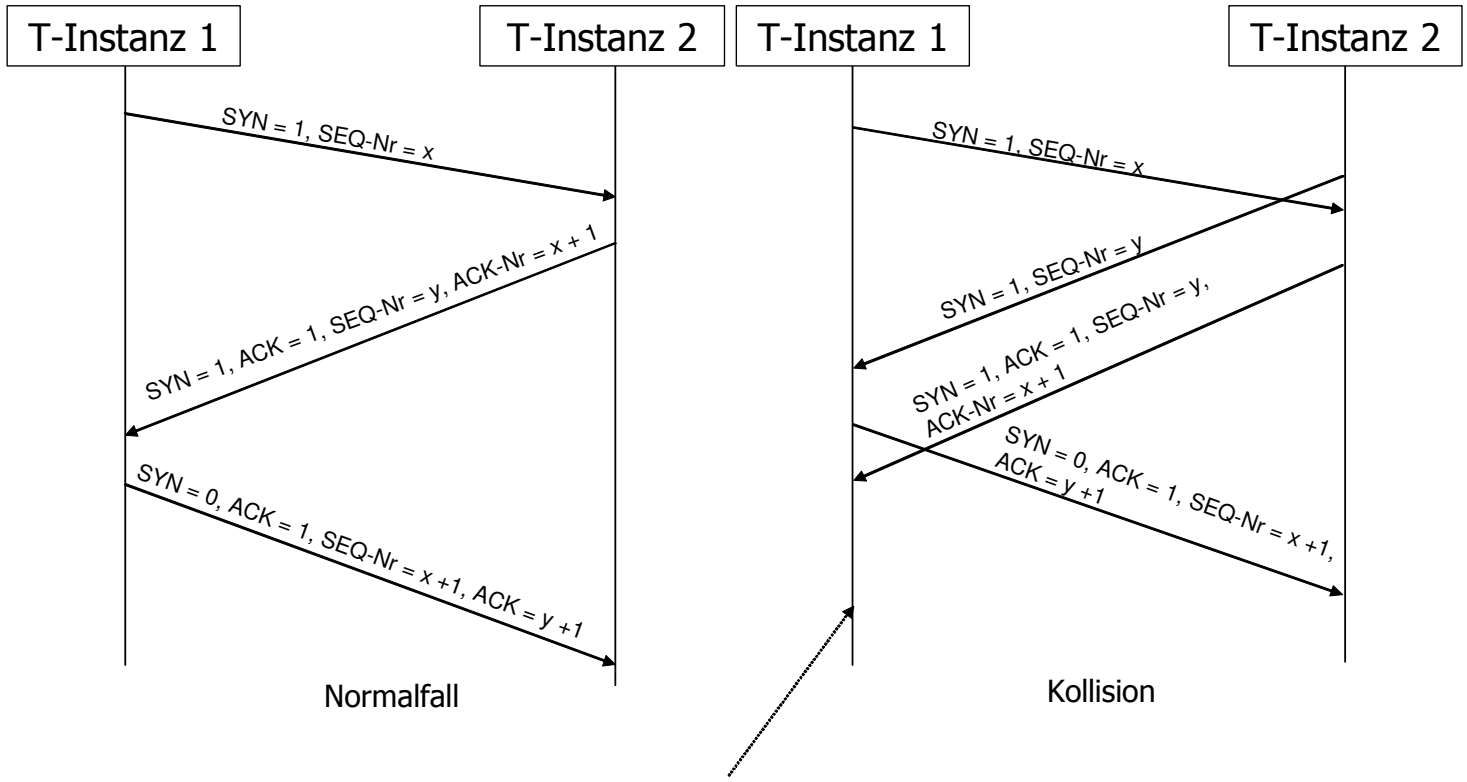
c_isn = Initial Sequence Number des Clients (Instanz 1)
s_isn = Initial Sequence Number des Servers (Instanz 2)

Verbindungsaufbau
Fehlerszenarien

- TCP muss mehrere Fehlersituationen richtig bearbeiten
  - Gleichzeitiger Verbindungsaufbauversuch beider Partner darf nur zu einer Verbindung führen
  - Alte Duplikate von TCP-Segmenten werden beim Verbindungsaufbau empfangen
    - Reset der Verbindung (RST-Bit) und erneuter Aufbau
  - Halb-offene Verbindung erkennen
    - Mit Reset (RST-Bit) abbauen
  - ...

Verbindungsaufbau
Kollisionsfall

- Normalfall und Kollision (gleichzeitiger Verbindungsaufbau) im Vergleich



Verbindungsaufbau
Kollision der Sequenznummern (1)

- **Problem:**
  - Erneuter, schneller Verbindungsaufbau nach Crash könnte zu Sequenznummern-Kollision führen
  - Ein noch altes TCP-Segment könnte bei neuer „Inkarnation" der Verbindung (gleiche Adressparameter) ankommen und nicht erkannt werden
  - Das muss verhindert werden

Verbindungsaufbau
Kollision der Sequenznummern (2)

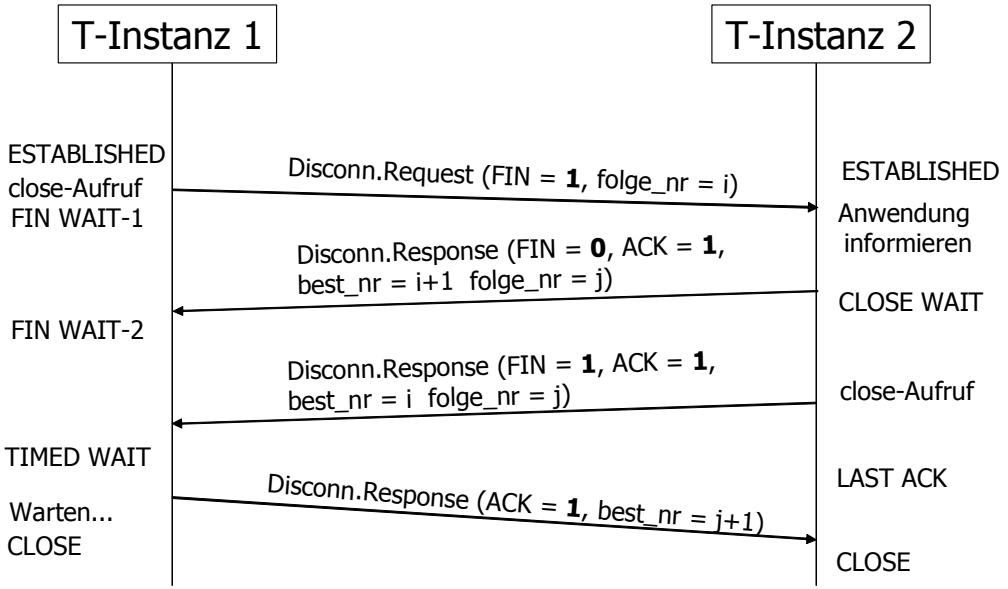
- **Lösung zur Synchronisation der Sequenznummern:**
  - Keine netzweit globale Uhr zur Bestimmung der Sequenznummer
  - **Keine feste Vorgabe**, wie ISN ermittelt wird → auch implementierungsabhängig
  - Generierung der ISNs anhand eines (fiktiven) Zeitgebers
    - Vorschlag im RFC 793, S. 28: Zyklus des Zeitgebers von 4,55 Stunden, Zeitgeber wird alle 4 ms erhöht
  - Einsatz des 3-Way-Handshake zum Synchronisieren der Sequenznummern beim Verbindungsaufbau
  - **Max. Segmentlebensdauer (MSL)** berücksichtigen: Wurde im RFC 792 auf 2 Minuten festgelegt
    - Diese Zeit muss gewartet werden, bevor nach einem Crash einer Verbindung eine neue ISN zugewiesen wird
    - Schwierig zu implementieren ohne persistente Speicherung der Verbindungsdaten!

Verbindungsabbau Protokoll (1)

- Verbindungsabbau-Protokoll:
  - Modifizierter Dreiwege-Handshake-Mechanismus
  - Jede der beiden Verbindungen der Vollduplex-Verbindung wird abgebaut, d.h. beide Seiten bauen ihre „Senderichtung" ab
- Ablauf:
  - Aktiv abbauender Partner sendet zunächst ein Segment mit **FIN=1**
  - Passiver Partner antwortet zunächst mit einem **ACK** und informiert die Anwendung (Signalisierung implementierungsabhängig!)
  - Wenn die Anwendung **close** aufruft, sendet die Partnerinstanz ebenfalls ein Segment mit **FIN=1**
  - Aktiver Partner sendet abschließend ein Segment mit **ACK=1**

Verbindungsabbau Protokoll (2)

- Client baut die Verbindung ab (auch Server kann es)
- Alle Segmente mit Folgenummer < i bzw. j sind noch zu verarbeiten
- FIN-Segment zählt Sequenznummer **um 1 hoch** (zählt als 1 Datenbyte)



Zustände im TCP-Zustandsautomat:
ESTABLISHED, FIN WAIT-1, FIN WAIT-2, TIMED WAIT, CLOSE, CLOSE WAIT, LAST ACK

Verbindungsabbau
Signalisierung beim passiven Partner

- Laut Zustandsautomat gibt es **mehrere Varianten** (genau 4) des Verbindungsabbaus
  - Es geht auch mit drei Segmenten (FIN=1 + ACK=1)
- Die **Signalisierung** des Verbindungsabbaus an die Anwendung ist der Implementierung überlassen und im RFC nicht genau beschrieben
  - Es kann eine Weile dauern, bis die Anwendung auf das close-Ereignis reagiert
  - Anwendung kann evtl. sogar eine Benutzereingabe erfordern
  - Dies hängt vom Programm ab
- Auch **abnormale Beendigung** einer Verbindung ist möglich
  - Segment mit **RST-Bit=1** wird gesendet und der Empfänger bricht die Verbindung sofort ab

Überblick

1. Einordnung und Aufgaben des Protokolls
2. Der TCP-Header
3. Verbindungsauf- und abbau
4. **Datenübertragung**

Tool zum Mitschneiden und Analysieren des Nachrichtenverkehrs

→ Ethereal/Wireshark-Sniffer

Sequenznummern und Quittierung

- Einsatz von Sequenznummern, die auf einzelnen Bytes, nicht auf TCP-Segmenten basieren (siehe Header->**Sequenznummer**)
- Die Sequenznummer enthält die Nummer des nächsten erwarteten Bytes
- Alle gesendeten Bytes werden vom Empfänger im Feld **Bestätigungsnummer kumulativ** quittiert
- Selektive, positive Quittierung auch möglich: Vorschlag in einem eigenen RFC

Sequenznummern und Quittierung

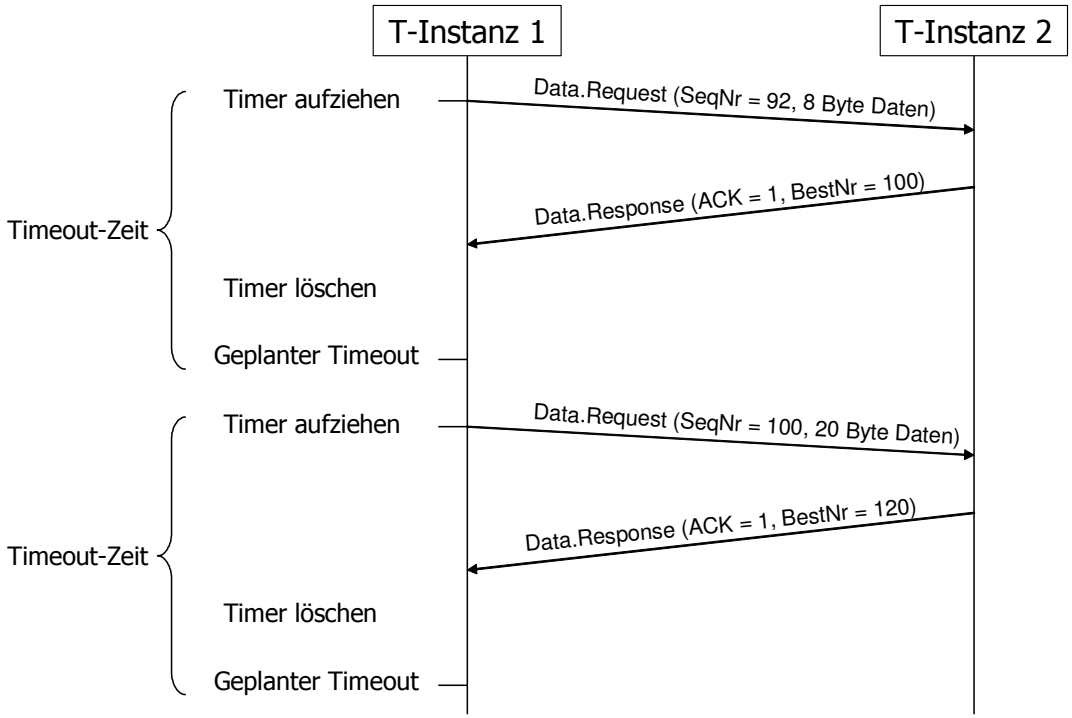
- Die Bestätigung muss von der empfangenden TCP-Instanz **nicht unbedingt sofort** gesendet werden, wenn noch Platz im Puffer ist
  - Hier besteht Implementierungsfreiheit für die Hersteller von TCP/IP-Stacks

Einschub:

- Sequenznummern-Anzahl bei TCP: 2^{32} (32-Bit-Feld)
- Vergabe mod 2^{32}
- Bei 64 Kbit/s kommt es frühestens nach 6,2 Tagen zu einer Wiederholung
- Bei 100 Mbit/s kommt es frühestens nach 340 s zu einer Wiederholung
- Bei 1 Gbit/s kommt es frühestens nach 34 s zu einer Wiederholung

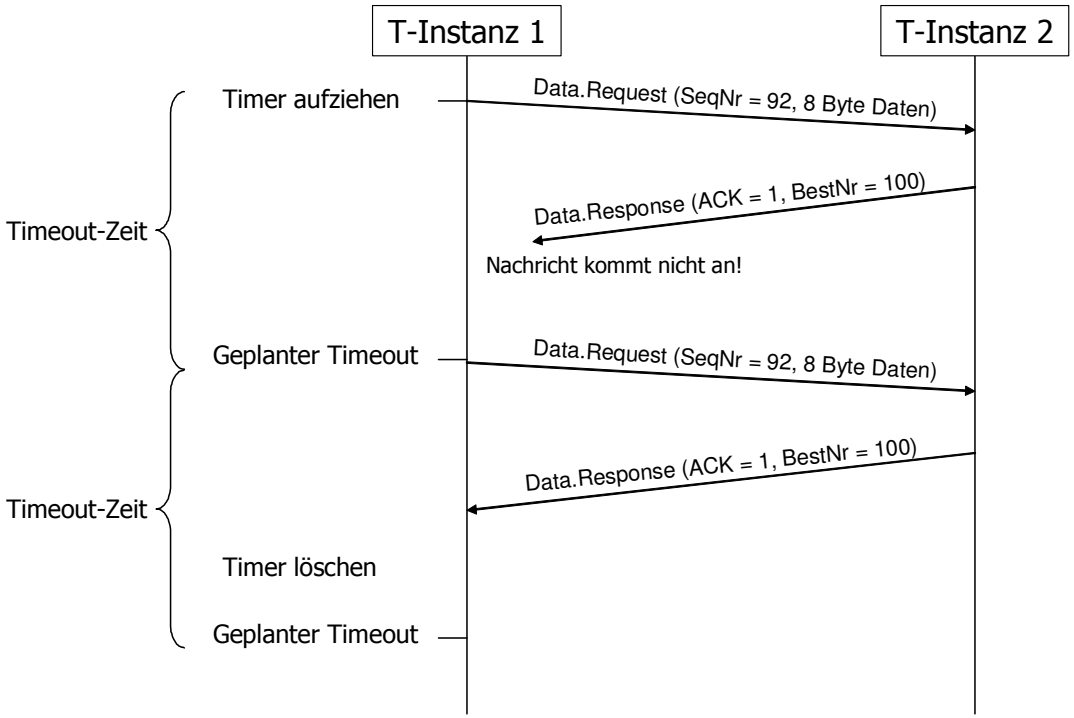
Szenario: Erfolgreiche Übertragung

- Sende-Instanz zieht Timer auf
- Timer wird nach ACK gelöscht



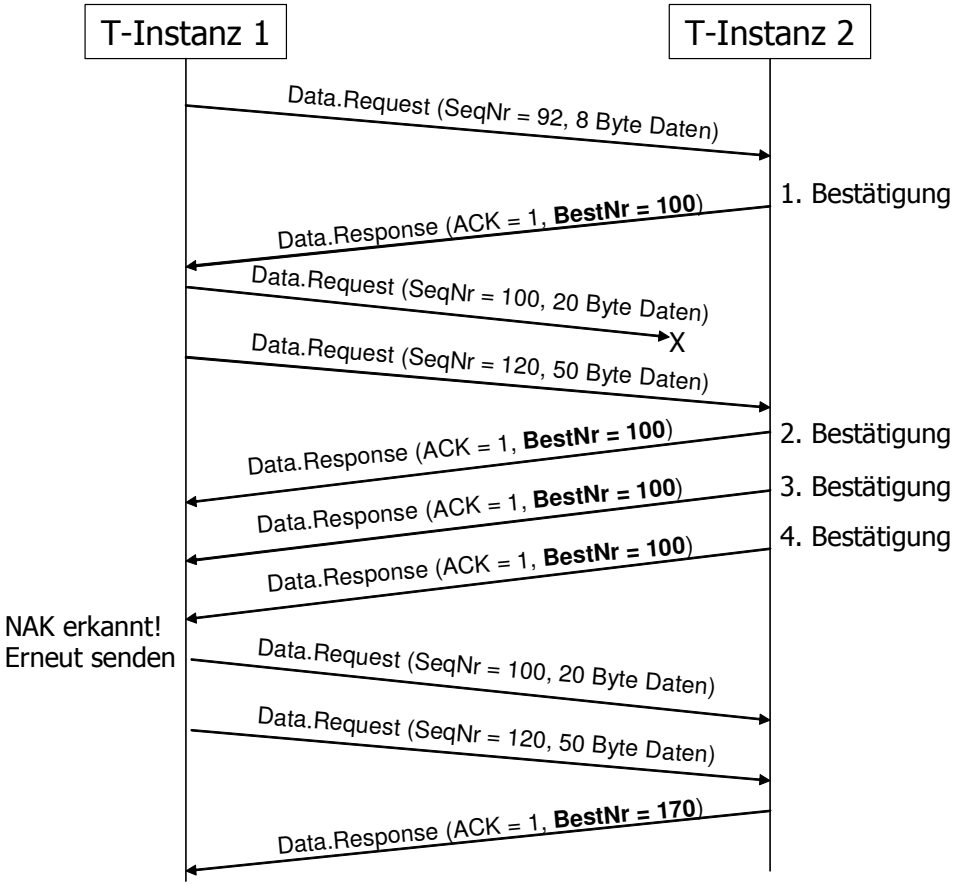
Szenario: Bestätigung geht verloren

- Sende-Instanz zieht Timer auf
- Timer läuft bei verlorengegangener Quittung ab
- TCP-Segment wird erneut gesendet



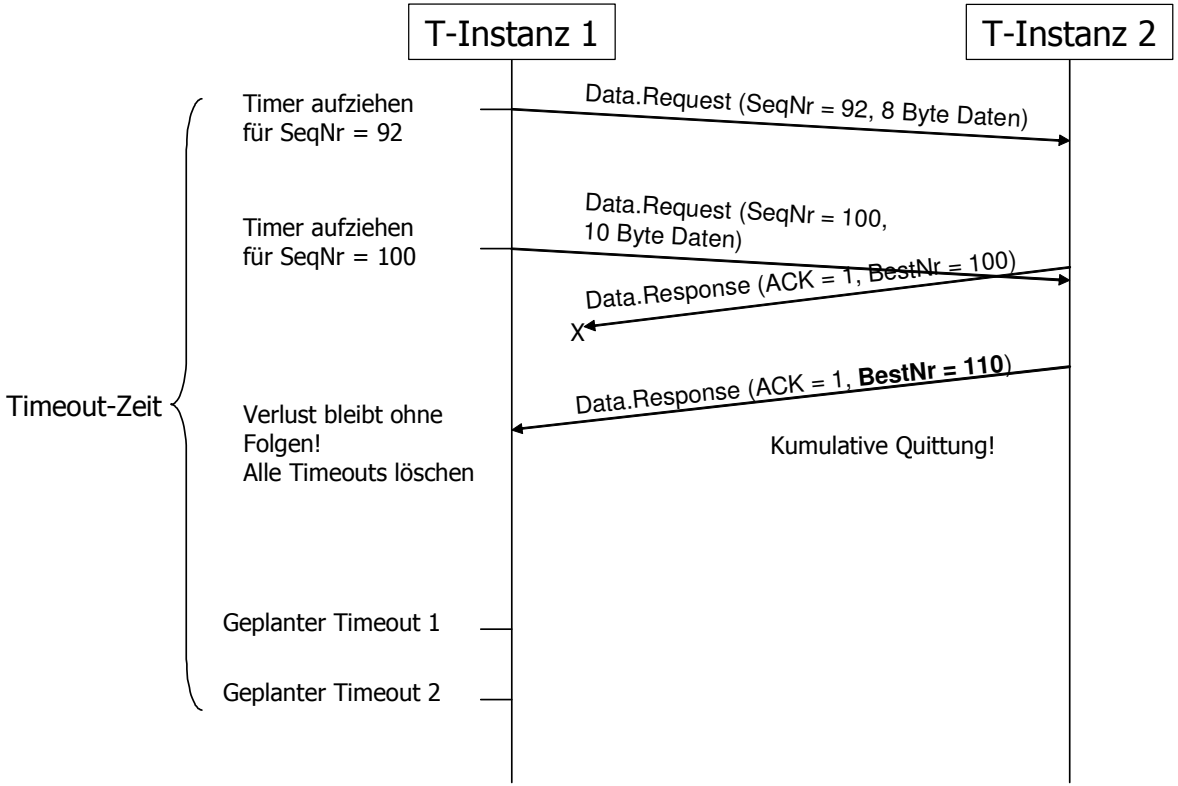
Szenario: Implizites NAK bewirkt Sendewiederholung

- Im Beispiel siehe 2. ACK wird **drei mal** vom Empfänger gesendet
- Sender erkennt Problem u sendet erneut → Problem wird vor dem Timerablauf erkannt
- Wird als **implizites NAK** bezeichnet!



Szenario: Kumulative Quittung verhindert erneutes Senden

- Verlust eines Segments bleibt ohne Folgen



Rückblick

1. Einordnung und Aufgaben des Protokolls
2. Der TCP-Header
3. Verbindungsauf- und abbau
4. Datenübertragung