Internetanwendungstechnik

Transportschicht

Gero Mühl

Technische Universität Berlin
Fakultät IV – Elektrotechnik und Informatik
Kommunikations- und Betriebssysteme (KBS)
Einsteinufer 17, Sekr. EN6, 10587 Berlin

Transportschicht

- Aufgaben der Transportschicht
  - Ende-zu-Ende-Protokoll zwischen Prozessen
  - Weitere Abstraktion von spezifischen Netzeigenschaften
  - Unterstützung mehrer Endpunkte auf einem Rechner durch Multiplexen und Demultiplexen mehrerer Transportbeziehungen → Ports
- Verbindungsorientierte Transportbeziehungen
  - Fehlerfreie Übertragung
  - Einhaltung der Reihenfolge
  - Keine Paketverluste und Duplikate
  - Flusssteuerung
- Verbindungslose Transportbeziehungen
  - Best Effort-Auslieferung von gesendeten Paketen

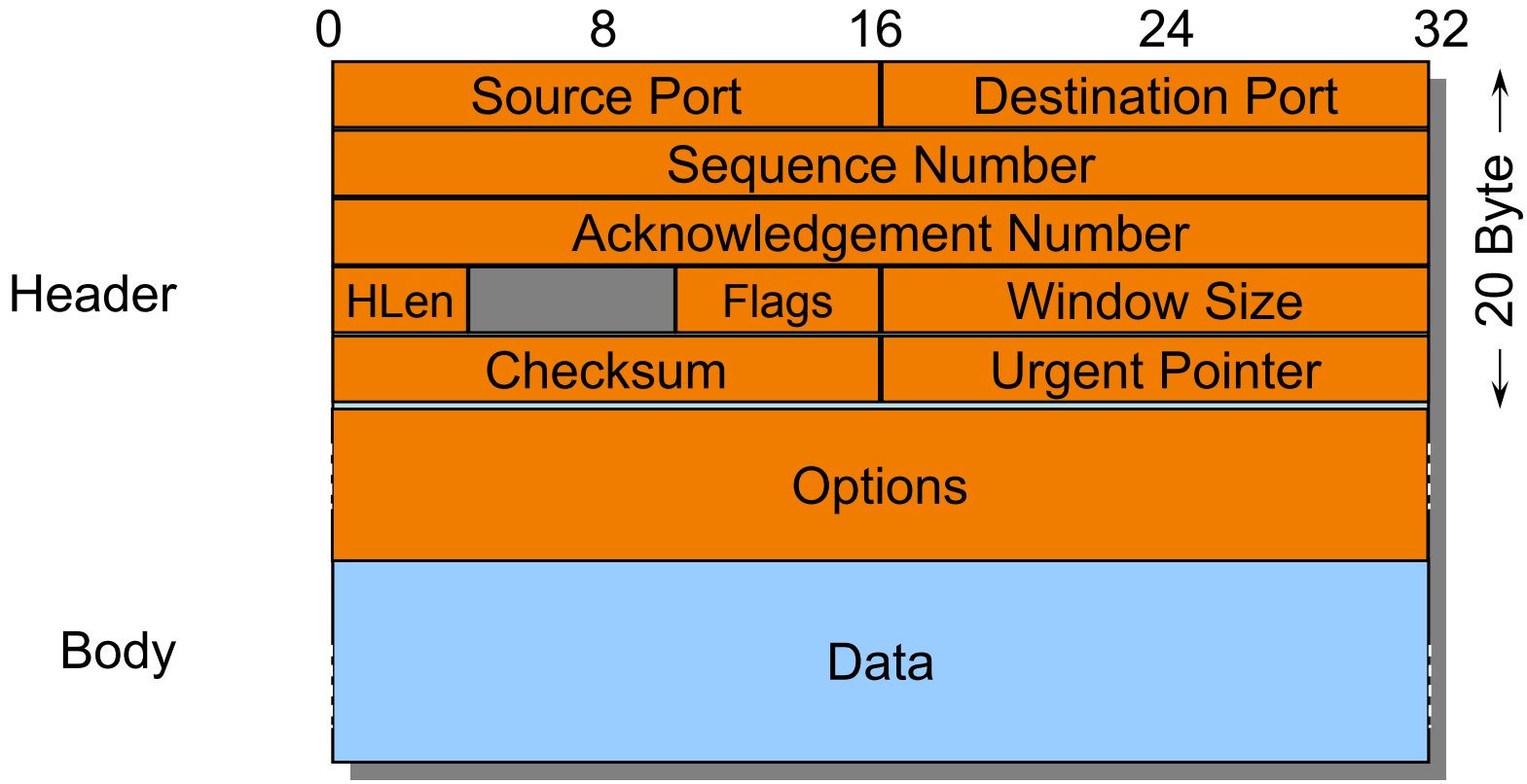
Transmission Control Protocol (TCP)

- RFCs 793, 1122, 1323
- Bereitstellung eines zuverlässigen bidirektionalen Bytestroms zwischen 2 Endpunkten in einem unzuverlässigen Netzverbund
  - Endpunkt: **IP-Adresse** + **TCP-Portnummer** (z.B. **130.1494.134**:**80**)
  - Unzuverlässiger Netzverbund: Nachrichten können verloren gehen, sich gegenseitig überholen und verfälscht werden
- Verbindungsorientierter Transportdienst
  - Teilt Anwendungsdaten in Blöcke (Segments) à max. 64 K Bytes (meistens ca. 1500 Bytes)
  - Jeder Block wird als ein IP-Paket versandt
  - **Sliding Window Protocol** zur *Fehlerbehandlung* und *Flusskontrolle*

Zuverlässiger Bytestrom

- Die Anwendungsdaten (Bytestrom) werden fehlerfrei empfangen, ohne Datenverluste oder -duplikate, in der Reihenfolge, in der sie gesendet wurden
- Dies wird durch das Versenden von Pakten mit Checksummen und Sequenznummern erreicht
- Die Anwendung selbst sieht keine Paketgrenzen
  - ein *send* kann zu mehreren *receives* führen und umgekehrt!
  - z.B. *send*(170 Bytes) + *send*(230) = *receive*(400)

Aufbau eines TCP-Pakets



Aufbau eines TCP-Pakets

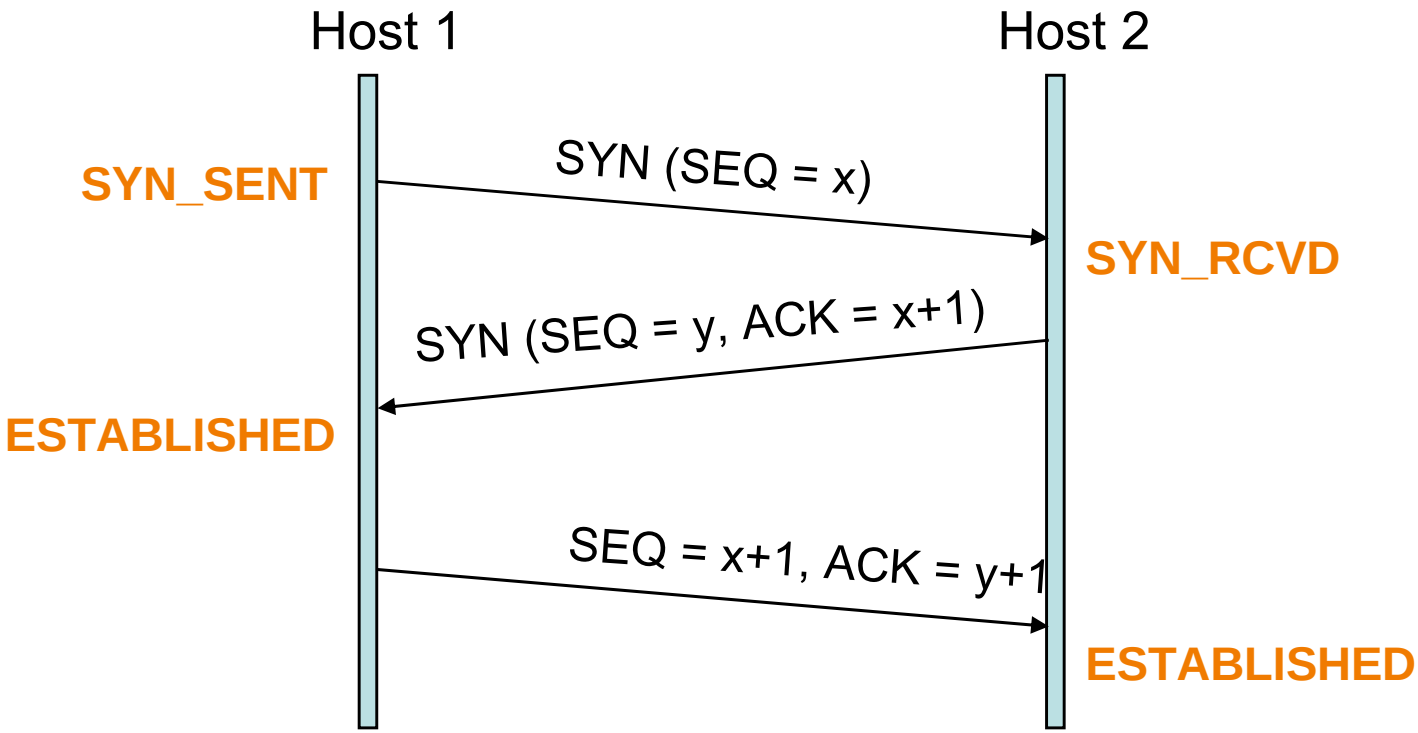
- Source- und Destination Port (Quell- und Ziel-Port)
  - Identifizieren die lokalen Endpunkte der Verbindung
- Sequence- und Acknowledgement Number (Folge- und Bestätigungsnummer)
  - Bytenummerierung und Empfangsbestätigung
- HLen (TCP Header Length, TCP Header-Länge)
  - Anzahl der 32-Bit-Wörter des Headers
- Flags
  - URG: dringende Daten liegen vor (siehe Urgent Pointer)
  - ACK: Bestätigung der Daten (siehe Acknowledgement Number)
  - PSH: Daten sofort an Anwendung weiterleiten (nicht puffern)
  - RST: Reset einer Verbindung
  - SYN: Signalisiert Verbindungsaufbau
  - FIN: Signalisiert Verbindungsabbau

Aufbau eines TCP-Pakets

- Window Size (Fenstergröße)
  - Größe des Empfangsfensters beim Sliding Window Protocol
  - Flusssteuerung
- Checksum (Prüfsumme)
  - Gebildet über TCP-Header + Daten + IP-Pseudo-Header
  - Berücksichtigt IP-Adressen aus Schicht 3!
- Urgent Pointer (Dringend-Zeiger)
  - Zeigt auf dringende Daten (vgl. Interrupt)
  - Erlaubt Verarbeitung außerhalb der Reihenfolge
- Options (Optionen)
  - n 32-Bit-Wörter
  - Aushandlung der TCP-Segmentgrößen
  - Aushandlung skalierter Fenstergrößen

TCP-Verbindungsaufbau

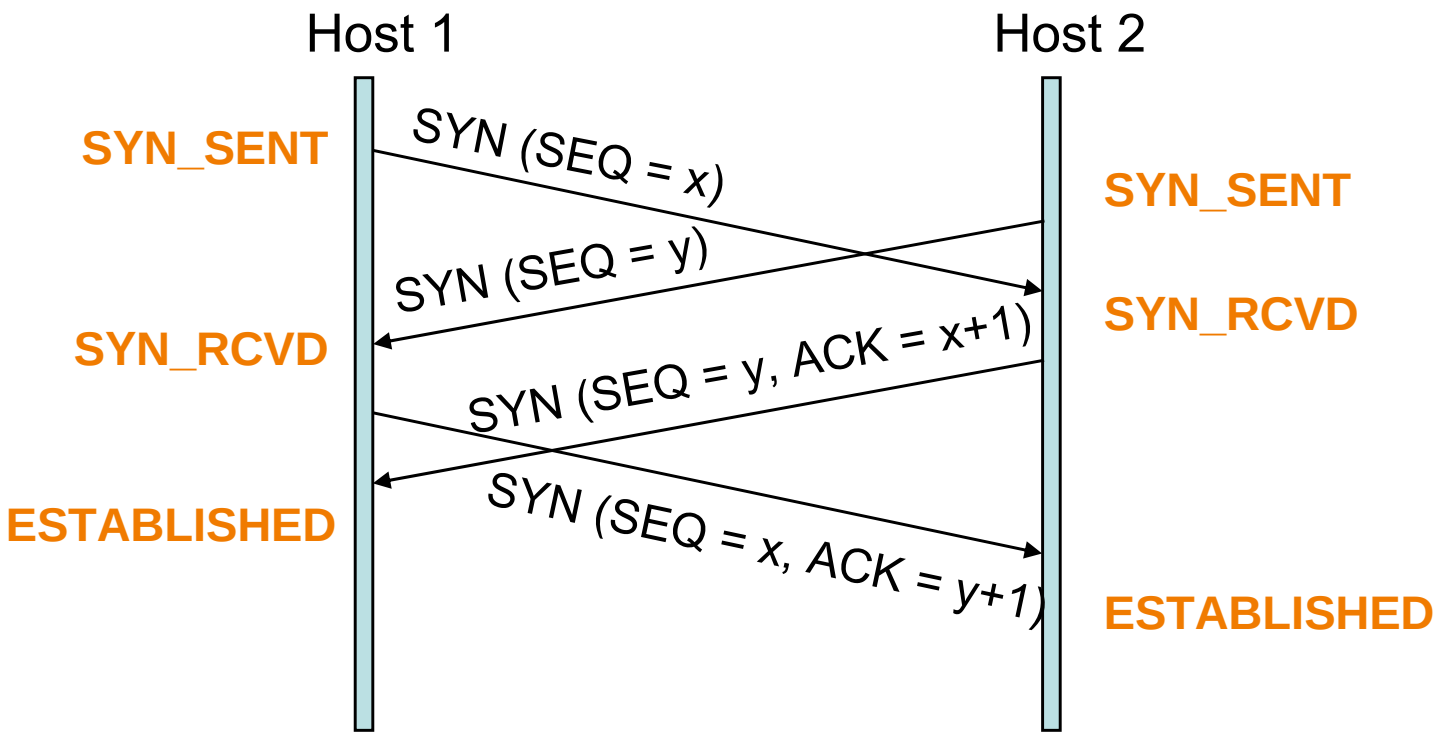
> Three-Way Handshake



> …oder Ablehnen mit „RST“!

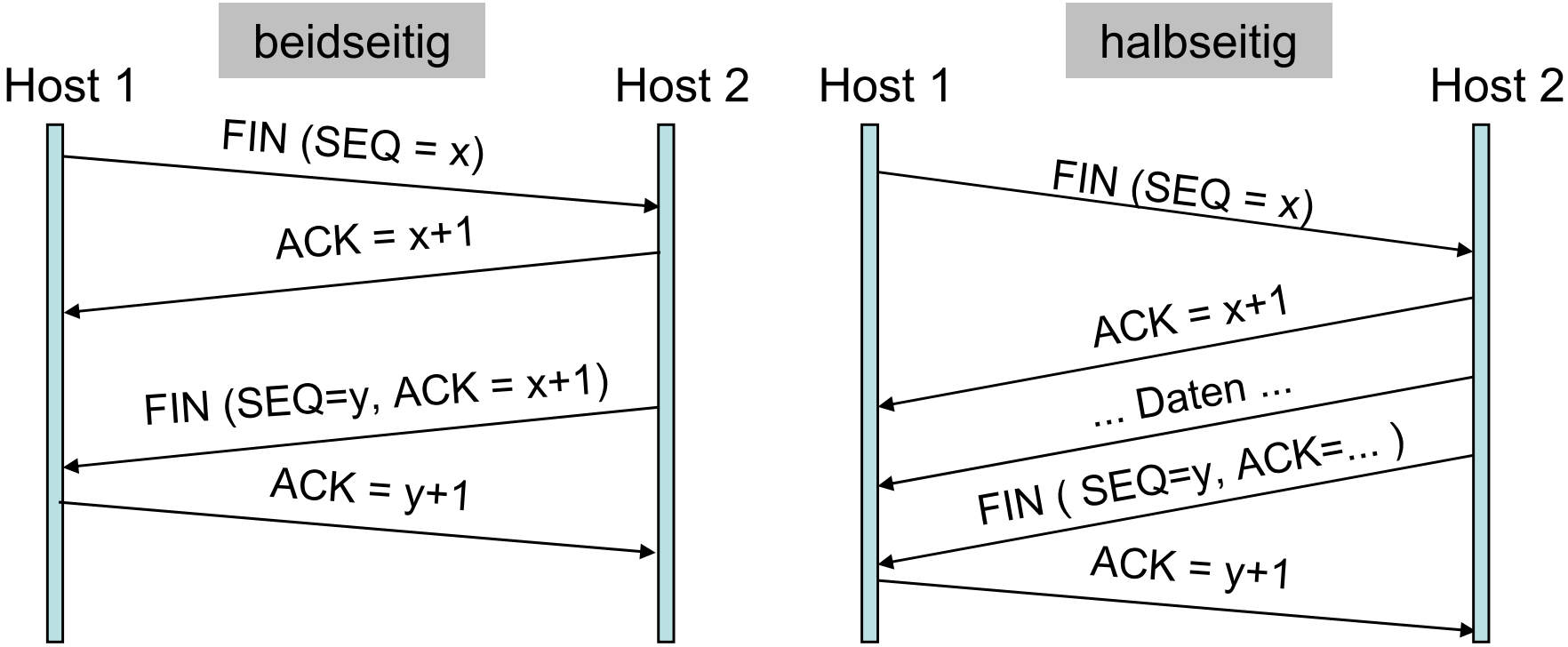
TCP-Verbindungsaufbau

> Simultane Eröffnung

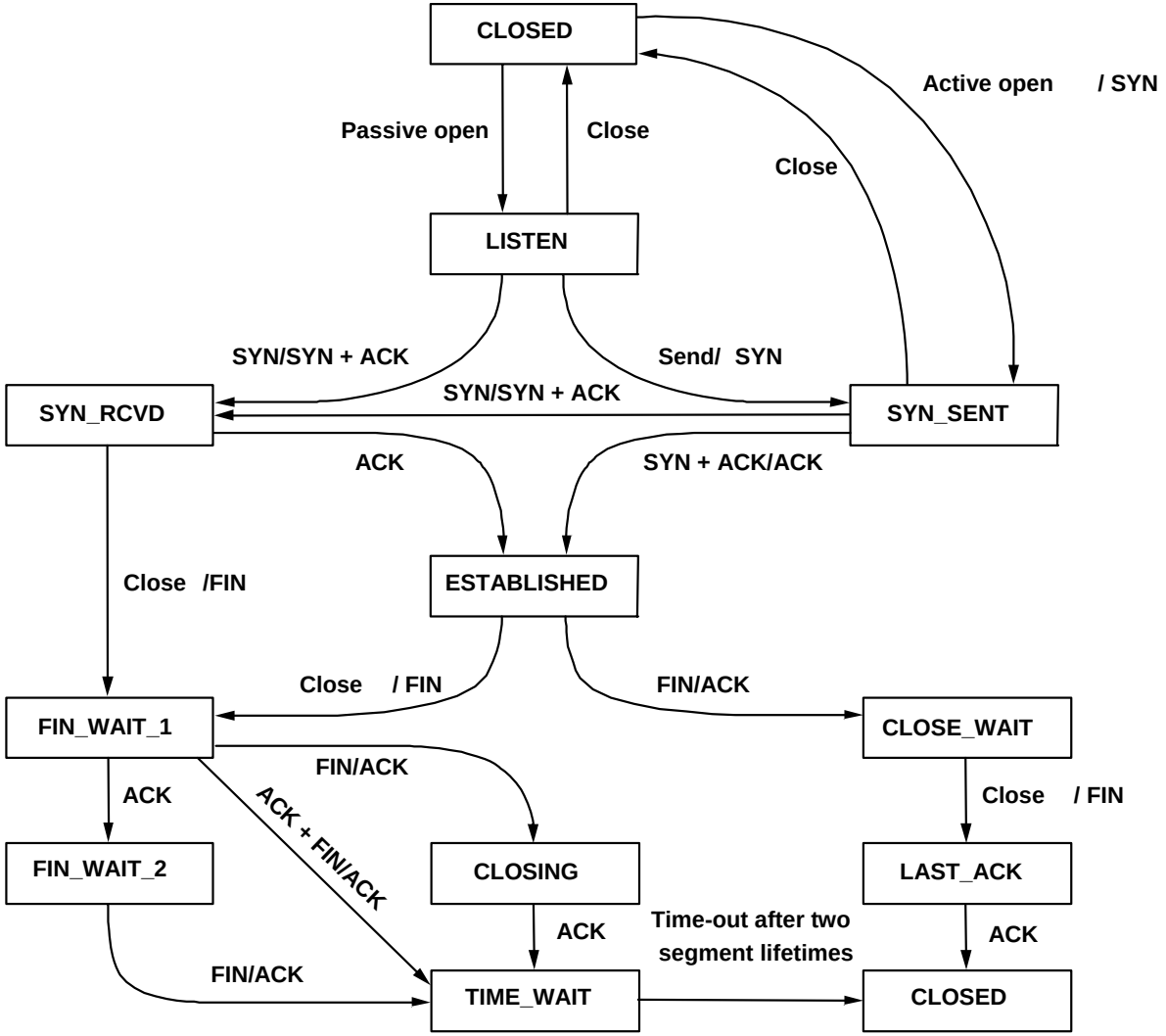


TCP-Verbindungsabbau

- Jede Partei kann ein Segment mit gesetztem „FIN“ schicken
- Bestätigung von FIN schließt die betroffene Richtung
- Timer für FIN-Segment



TCP-Verbindungsmanagement



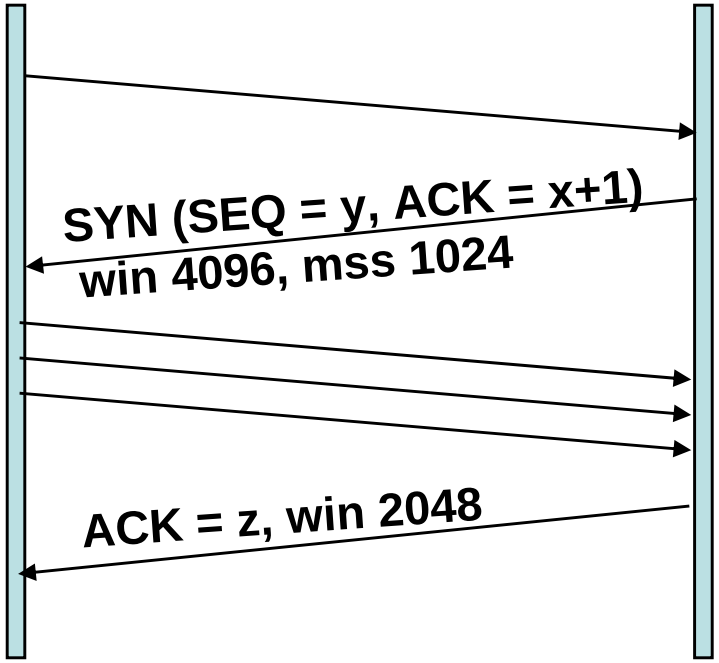
TCP-Zustandsautomat

TCP-Segmente

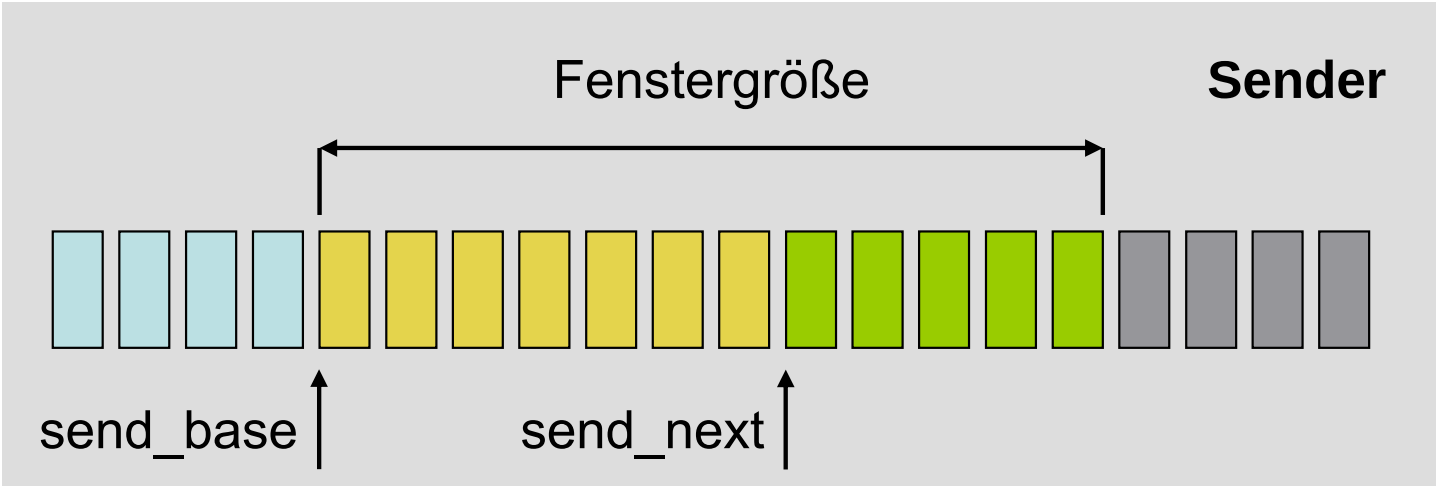
- Jede Seite einer Verbindung kann beim „SYN“ ihre *maximale Segmentgröße (MSS)* bekannt geben
  - Größere Segmente sind effizienter (Warum?)
  - MSS ≤ MTU des Netz – IP-Header (20 Byte) – TCP-Header (20 Byte)
  - Z.B. Ethernet MTU = 1500 Byte → MSS = 1460 Byte
  - Z.B. Berkeley UNIX Implementierung → MSS = 1024 Byte
  - Voreinstellung MSS = 536 Byte
    - 536 Byte Daten + TCP Header (20) + IP Header (20) = minimale Maximalgröße eines IP-Pakets (576 Byte)

Empfängerfenster

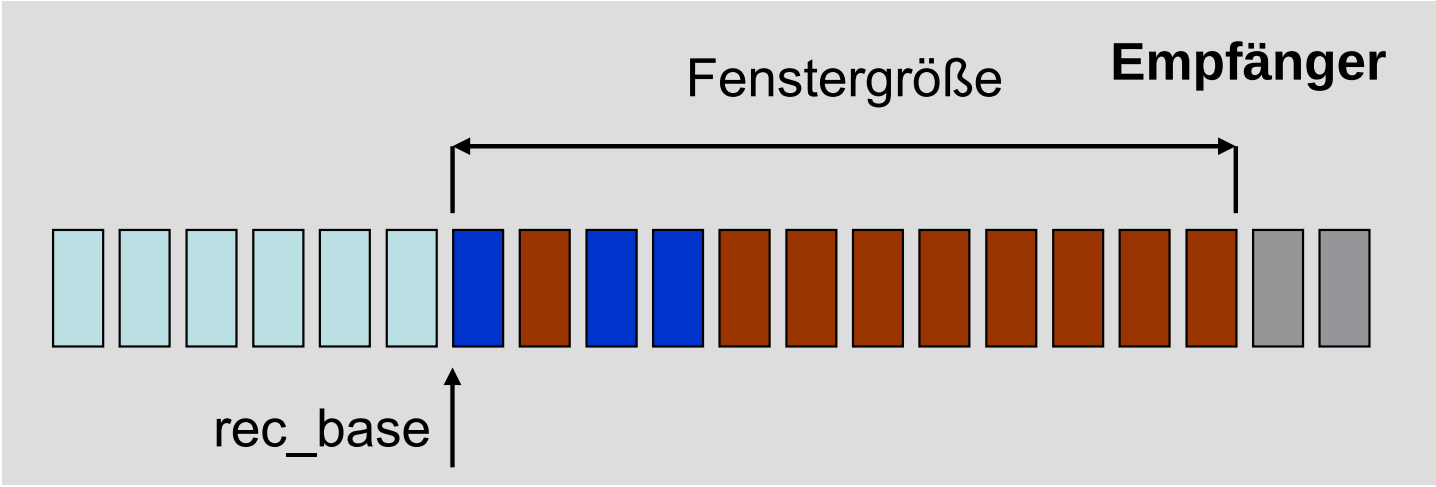
- Angabe der Größe des Empfängerfensters bei Bestätigungen
- Entspricht der Anzahl an Bytes, die der Empfänger bereit ist, max. als nächstes zu akzeptieren
- Ziel: Überlastungsschutz → Flow Control
- 16 Bit Größe → max. 64KB (Skalierungsfaktor im Header möglich)



Sliding Window Protocol



- gesendet und bestätigt
- gesendet aber nicht bestätigt
- kann gesendet werden
- kann noch nicht gesendet werden



- ausgeliefert an Anwendung
- empfangen und bestätigt
- erwartet
- Kann noch nicht empfangen werden

TCP–Effizienz: Senderseite

- Tinygrams
  - Eine Anwendung sendet ständig kleinste Datenmengen (z.B. 1 Byte) → enormes Verkehrsaufkommen
  - 1 Byte Daten + 20 Byte TCP Header + 20 Byte IP Header = 41 Byte Übertragungslast für 1 Byte Nutzlast
  - Zusätzlicher Aufwand durch z.B. darunter liegendes Ethernet (14 Bytes Header + 4 Bytes Checksumme)
- Lösung: Nagle-Algorithmus
  - Leistungsverbesserung durch Sammeln von Sendedaten
  - Allerdings: Bei interaktiven Anwendungen (z.B. SSH, X Windows) muss Nagle‘s Algorithmus deaktiviert werden

TCP–Effizienz: Empfängerseite

- Silly Window Syndrome
  - Nur kleine Segmente werden geschickt, obwohl der Sendepuffer genügend Daten für ein größeres Segment enthält
- Ursache
  - Anwendung beim Empfänger liest Daten, z.B. byteweise
  - Speicher im vollen Puffer wird byteweise frei
  - Empfänger bietet zu kleines Empfängerfenster (hier 1 Byte) an, anstatt zu warten, bis ein größeres Fenster möglich wäre
- Lösung: Clark-Algorithmus
  - Empfänger aktualisiert das Fenster nur, wenn Erhöhung um definierte Mindestmenge möglich ist

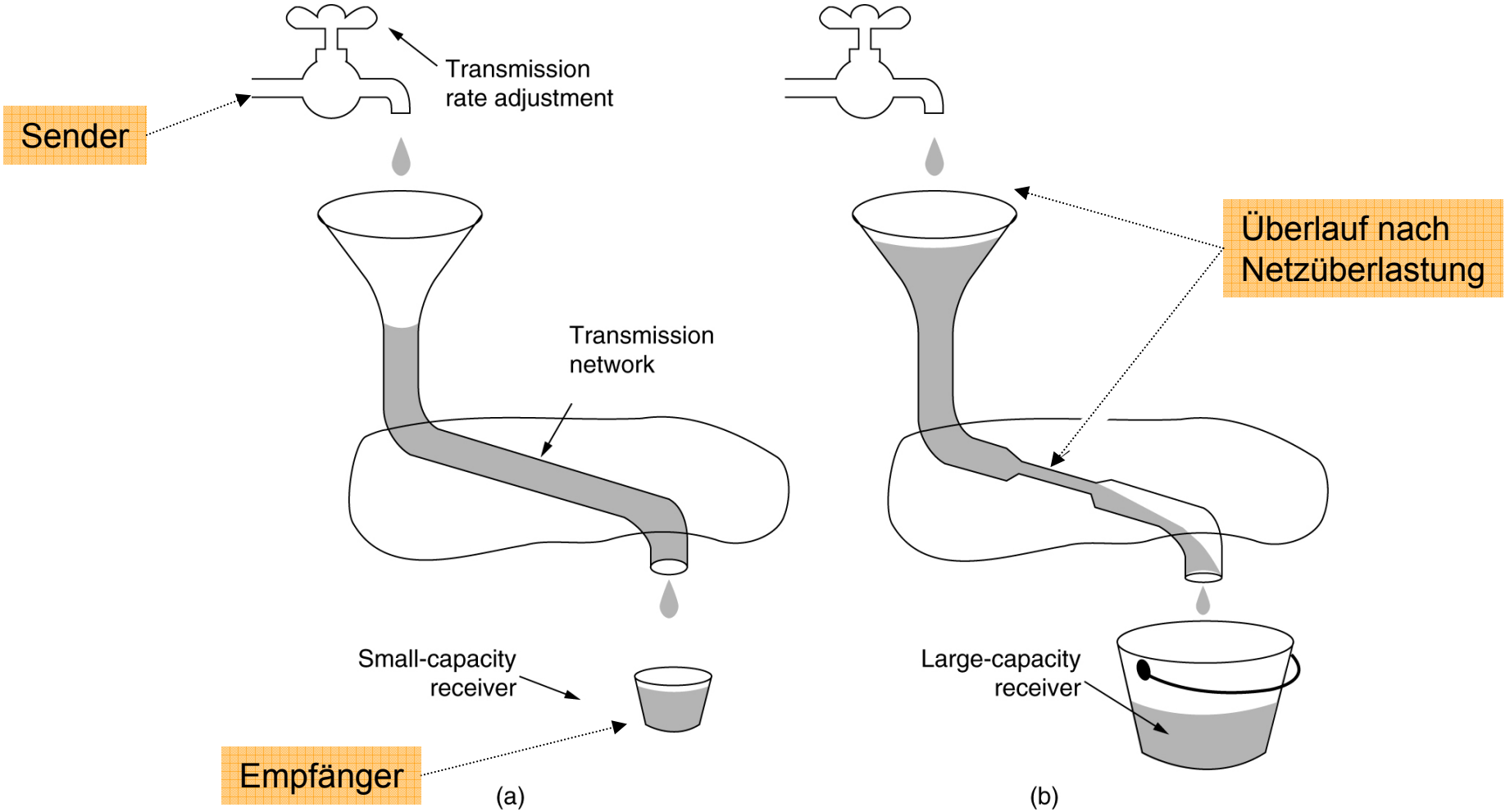
TCP–Effizienz: Empfängerseite

- TCP bestätigt außer der Reihe empfangene Segmente erst, wenn alle vorhergehende eingetroffen sind
- Beispiel:
  - Die Segmente 0, 1, 3, 4, 5, 6 kommen an (→ Segment 2 fehlt)
  - Es werden daher nur die Segmente 0 und 1 bestätigt
  - Die Segmente 3 bis 6 können beibehalten oder verworfen werden
  - Werden sie beibehalten und das fehlende Segment 2 trifft ein, so wird direkt Segment 6 bestätigt
  - Trifft das ACK nicht rechtzeitig ein, so beginnt der Sender erneut bei Segment 2 zu senden → ineffizient, weil nur ein Paket fehlt
- Lösung: NAK-Option [RFC 1106]
  - *Selektive Wiederholung* eines fehlenden Segments anstatt „*go back n*“-Protokoll
  - Empfänger schickt *NAK* (= ***N****egative* ***A****c****K****nowledgement*) mit Segmentnummer

TCP-Überlastkontrolle

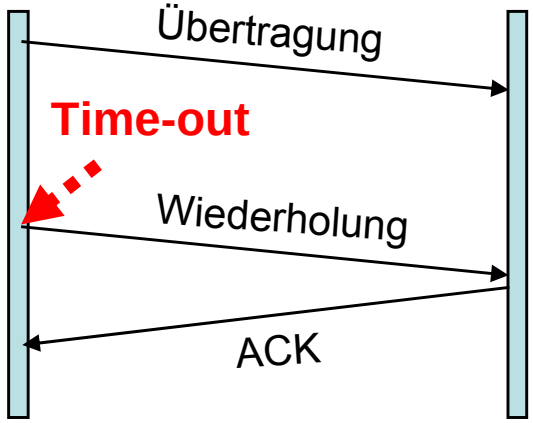
- TCP führt Überlastkontrolle aus (engl.: Flow Control)
- Durch Überlast entsteht Paketverlust
  - Mangelnde Empfängerkapazität (z.B. Puffer)
  - Überlastete Router verwerfen Pakete
- Vermeiden der Überlastungen
  - Zusätzlich zum Empfängerfenster ein Überlastungsfenster beim Sender verwendet
  - Gesendet wird dann immer das Minimum von Empfänger- *und* Überlastungsfenster
  - ⇒ Dynamische Anpassung des Volumens der gesendeten Daten

TCP-Überlastkontrolle



TCP-Timer-Management

- Timer/Time-outs dienen der Überwachung der Verbindung
- Problem: Wahl der Länge des Time-Outs?
  - Z.B. Retransmission Timer (Bestimmt die Wartezeit auf eine Bestätigung)
  - Time-out zu kurz → unnötige Wiederholung
  - Time-out zu lang → unnötige Verzögerung von Wiederholungen
  - Idee: Orientierung an der Round Trip Time



- Round Trip Time („Rundreisezeit“)
  - Zeit für eine Nachricht vom Sender zum Empfänger und zurück
  - Sich (evtl. schnell) ändernder Parameter
- Dynamische Algorithmen zur Anpassung des Time-out-Intervalls (Jacobsen, 1988)

Jacobsen Algorithmus

- Round Trip Time (RTT) sowie Schwankungen der RTT zur Wahl des Time-outs heranziehen
- Schätzung der Round Trip Time (RTT)

$$RTT = \alpha_1 \cdot RTT + (1-\alpha_1) \cdot M$$

- Berechnung der durchschnittlichen Abweichung (D)

$$D = \alpha_2 \cdot D + (1-\alpha_2) \cdot |RTT-M|$$

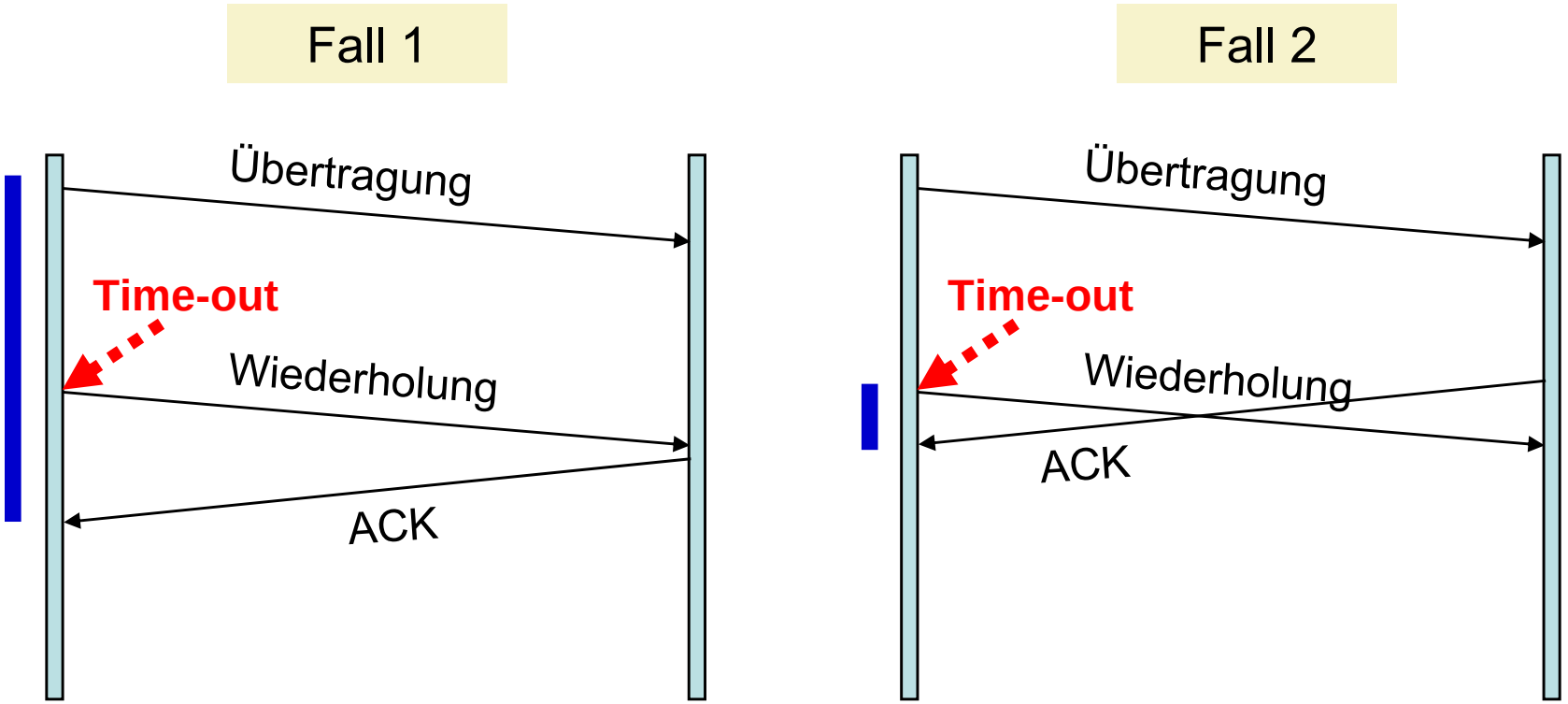
- Wahl des Timeouts

$$Timeout = RTT + 4 \cdot D$$

M: RTT des letzten bestätigten Segments
α_1, α_2: Glättungsfaktoren (typisch $\alpha_1 = \alpha_2 = 7/8$)

Karn-Algorithmus

> Mögliche falsche Zuordnung von Timer und Bestätigung?



Karn-Algorithmus

- Wenn eine Bestätigung nach einer Wiederholung ankommt, tritt folgendes Problem auf
  - War die Bestätigung für die ursprüngliche Übertragung oder für die Wiederholung?
  - Bei Missinterpretation könnte der RTT fälschlicherweise auf einen viel zu kleinen Wert aktualisiert werden!

- Lösung: Karn-Algorithmus
  - Time-out nach einer Wiederholung weiterhin verdoppeln (bzw. anpassen), bis ein Segment beim ersten Versuch erfolgreich übertragen wird

Persistence Timer

- Verlorene Ack-Nachrichten können zu Missverständnissen zwischen Sender und Empfänger führen
- Es kann dann zu einem Verklemmungszustand kommen (jeder wartet auf den anderen)
- Lösung
  - Periodische Nachfragen gesteuert durch den Persistence Timer

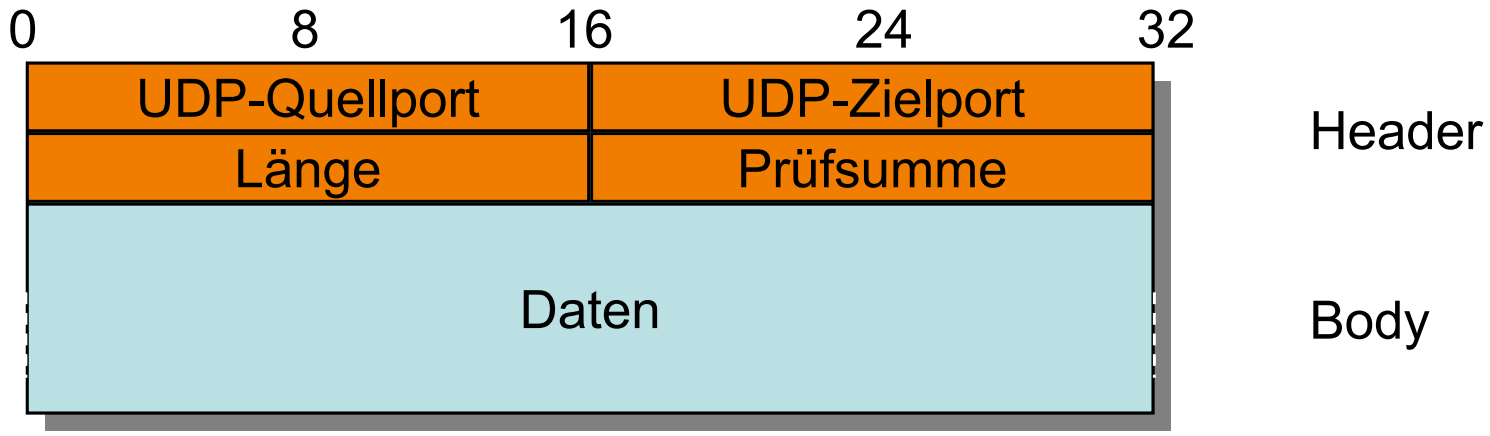
Keep Alive-Timer

- Nicht Teil des Standards, kontrovers diskutiert
- Idee Test-Pakete senden, wenn Verbindung längere Zeit inaktiv, um „tote“ Verbindungen zu entdecken
- Beide Seiten können prüfen, ob der andere noch da ist
  → falls keine Antwort, Verbindung beenden
- Verursacht zusätzliche Netzlast
- „Gesunde“ Verbindungen können beendet werden, wenn das Netz vorübergehend getrennt ist

User Datagram Protocol (UDP)

- Verbindungsloser Transportdienst [RFC 768]
- Ermöglicht das Senden einzelner Pakete an einen Endpunkt (IP-Adresse + UDP-Portnummer)
- Empfänger empfängt einzelne Pakete
- Pakete können verloren gehen und in beliebiger Reihenfolge beim Empfänger eingehen
- Bietet größtmögliche Freiheit und Flexibilität bei weniger Overhead als bei TCP
- Die Anwendung kümmert sich selbst um Fehlerbehandlung und Flusssteuerung (falls notwendig)
- Unterstützt auch Multicast-Kommunikation mittels IP-Multicast

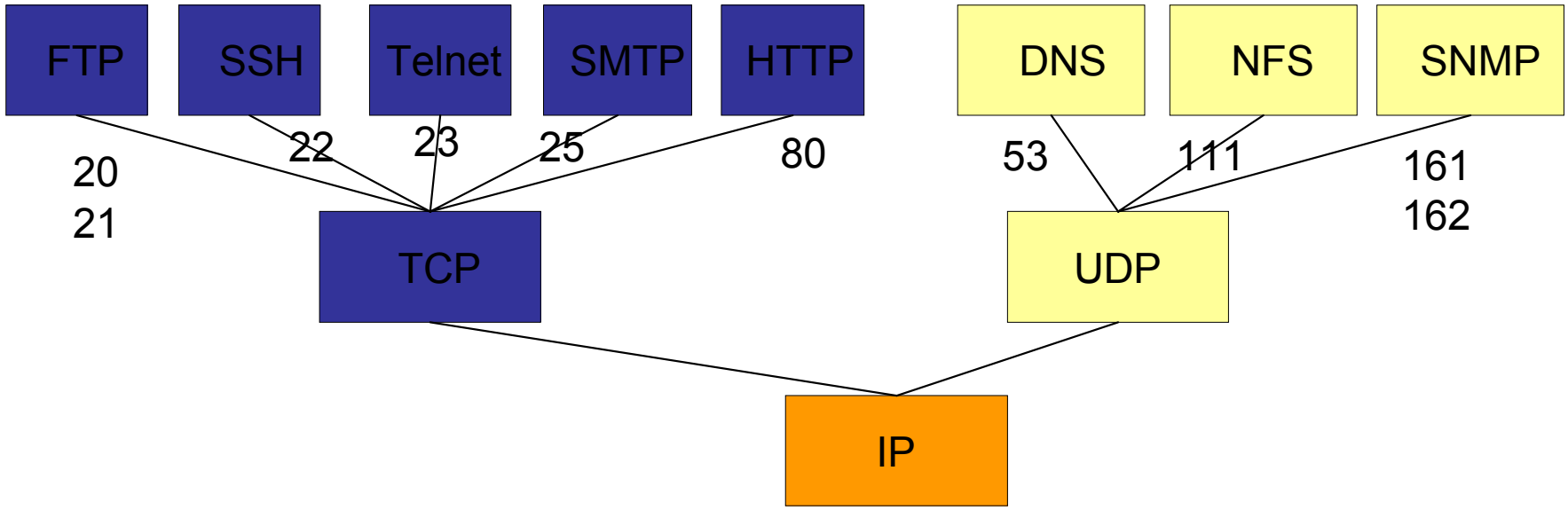
Aufbau eines UDP Pakets



Quell- und Zielport	wie bei TCP
Länge	Länge des Datagramms in Bytes (inkl. 8 Byte Header)
Prüfsumme	Fehlerüberprüfung für Header, Daten und wichtige IP-Informationen

Well-Known TCP/UDP-Ports

- Well-Known Ports (Portnummern < 1024) reserviert für bestimmte Dienste [http://www.iana.org/assignments/port-numbers]
- TCP- und UDP-Portnummern sind unabhängig voneinander



Literatur

1. V. Jacobson. *Congestion Avoidance and Control*. In Symposium Proceedings on Communications Architectures and Protocols, pages 314–329, Stanford, CA, USA, 1988. ACM Press.

2. W. R. Stevens. *TCP/IP Illustrated, Volume 1: The Protocols*. Addison-Wesley, 1991.

3. G. R. Wright and W. R. Stevens. *TCP/IP Illustrated, Volume 2: The Implementation*. Addison-Wesley, 1995.

4. J. Nagle. *Congestion Control in IP/TCP Internetworks*. RFC 896, January 1974.

5. D. D. Clark. *Window and Acknowledgement Strategy in TCP*. RFC 813, July 1982.

Fragen?