

Systeme II

5. Die Transportschicht

Thomas Janson[°], Kristof Van Laerhoven*, Christian Ortolf[°]

Folien: Christian Schindelbauer[°]

Technische Fakultät

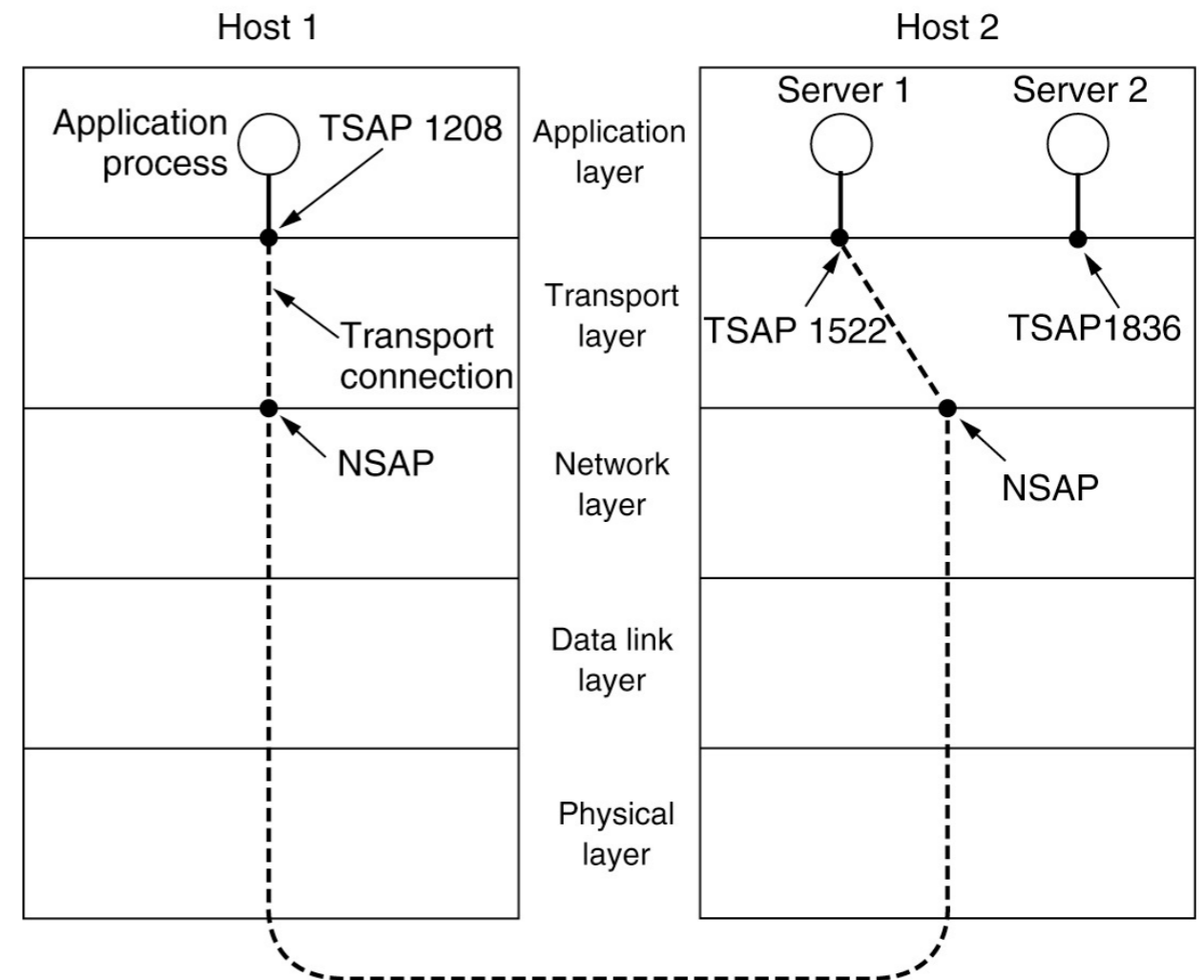
[°]: Rechnernetze und Telematik, *: Eingebettete Systeme

Albert-Ludwigs-Universität Freiburg

Version 24.04.2015

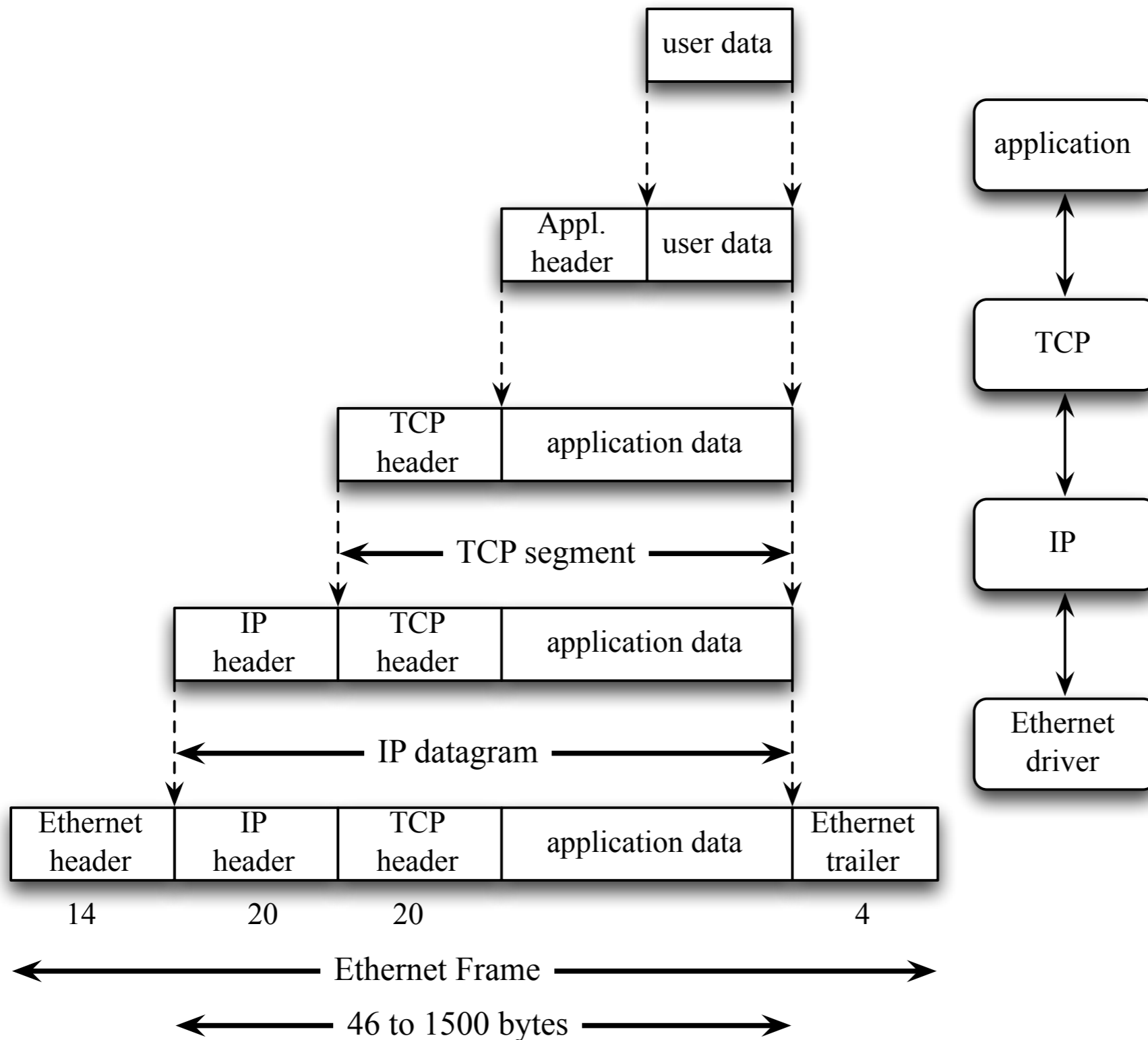
- Verbindungslos oder Verbindungsorientiert
 - Beachte: Sitzungsschicht im ISO/OSI-Protokoll
- Zuverlässig oder unzuverlässig
 - Best effort oder Quality of Service
 - Fehlerkontrolle
- Mit oder ohne Congestion Control
- Möglichkeit verschiedener Punkt-zu-Punktverbindungen
 - Stichwort: Demultiplexen
- Interaktionsmodelle
 - Byte-Strom, Nachrichten, „Remote Procedure Call“

- Die Netzwerkschicht leitet Daten an die Transportschicht unkontrolliert weiter
- Die Transportschicht muss sie den verschiedenen Anwendungen zuordnen:
 - z.B. Web, Mail, FTP, ssh, ...
 - In TCP/UDP durch Port-Nummern
 - z.B. Port 80 für Web-Server



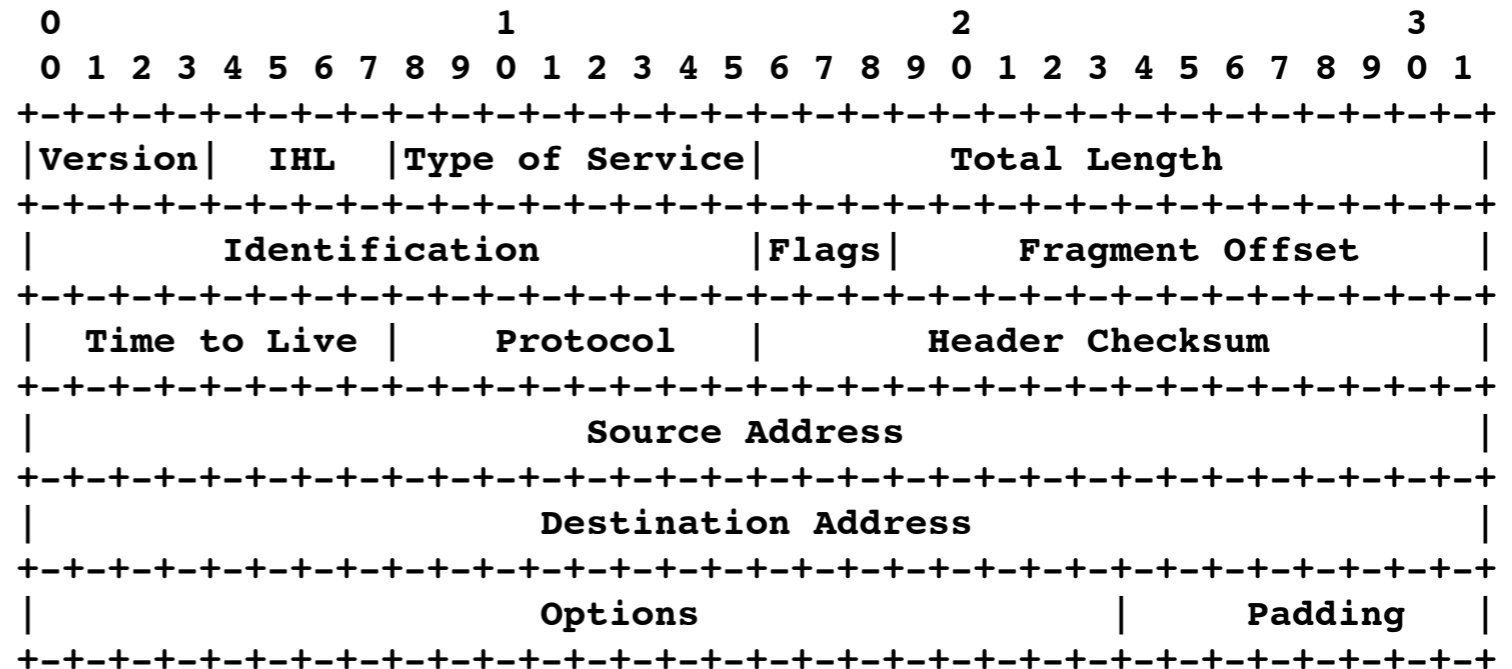
Multiplex in der Transportschicht

Port	Beschreibung	TCP	UDP
7	Echo	Ja	Ja
20, 21	FTP – Datenübertragung, Kontrolle	Ja	Nein
23	Telnet	Ja	Nein
25	Simple Mail Transfer Protocol (SMTP)	Ja	Nein
37	TIME	Ja	Ja
53	Domain Name System (DNS)	Ja	Ja
69	Trivial File Transfer Protocol (TFTP)	Nein	Ja
80	HTTP	Ja	Nein
110	POP-3	Ja	Nein
143	IMAP	Ja	Ja
443	HTTPS	Ja	Nein
543	RTSP	Ja	Nein
631	Internet Printing Protocol (IPP)	Ja	Ja



IP-Header (RFC 791)

- Version: 4 = IPv4
- IHL: Headerlänge
 - in 32 Bit-Wörter (>5)
- Type of Service
 - Optimiere delay, throughput, reliability, monetary cost
- Checksum (nur für IP-Header)
- Source and destination IP-address
- Protocol, identifiziert passendes Protokoll
 - Z.B. TCP, UDP, ICMP, IGMP
- Time to Live:
 - maximale Anzahl Hops

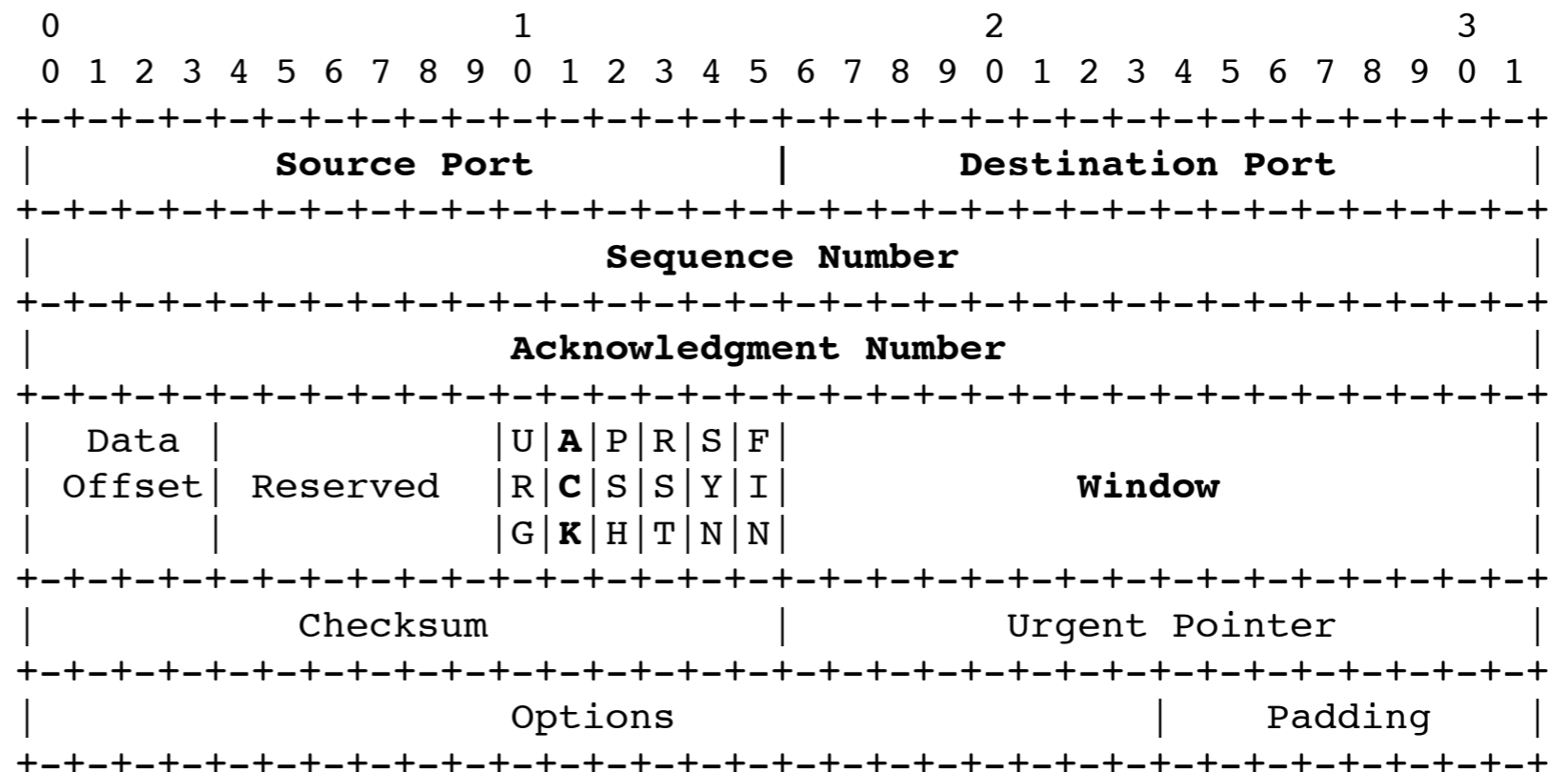


- Sequenznummer
 - Nummer des ersten Bytes im Segment
 - Jedes Datenbyte ist nummeriert modulo 2^{32}
- Bestätigungsnummer
 - Aktiviert durch ACK-Flag
 - Nummer des nächsten noch nicht bearbeiteten Datenbytes
 - = letzte Sequenznummer + letzte Datenmenge:

- Port-Adressen
 - Für parallele TCP-Verbindungen
 - Ziel-Port-Nr.
 - Absender-Port

- Headerlänge
 - data offset

- Prüfsumme
 - Für Header und Daten



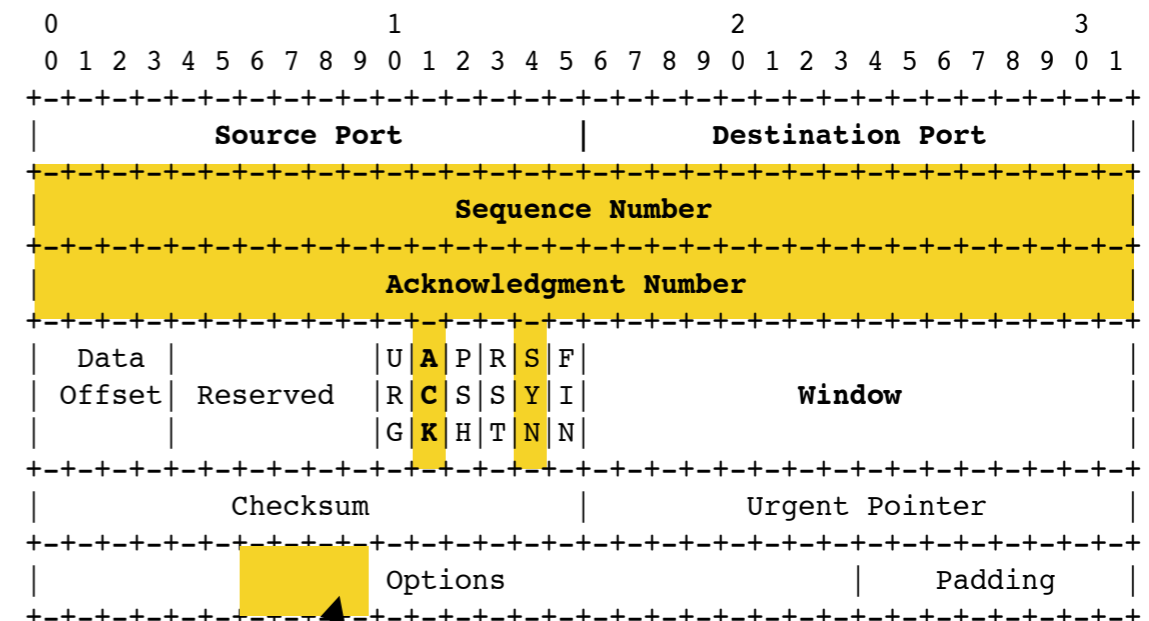
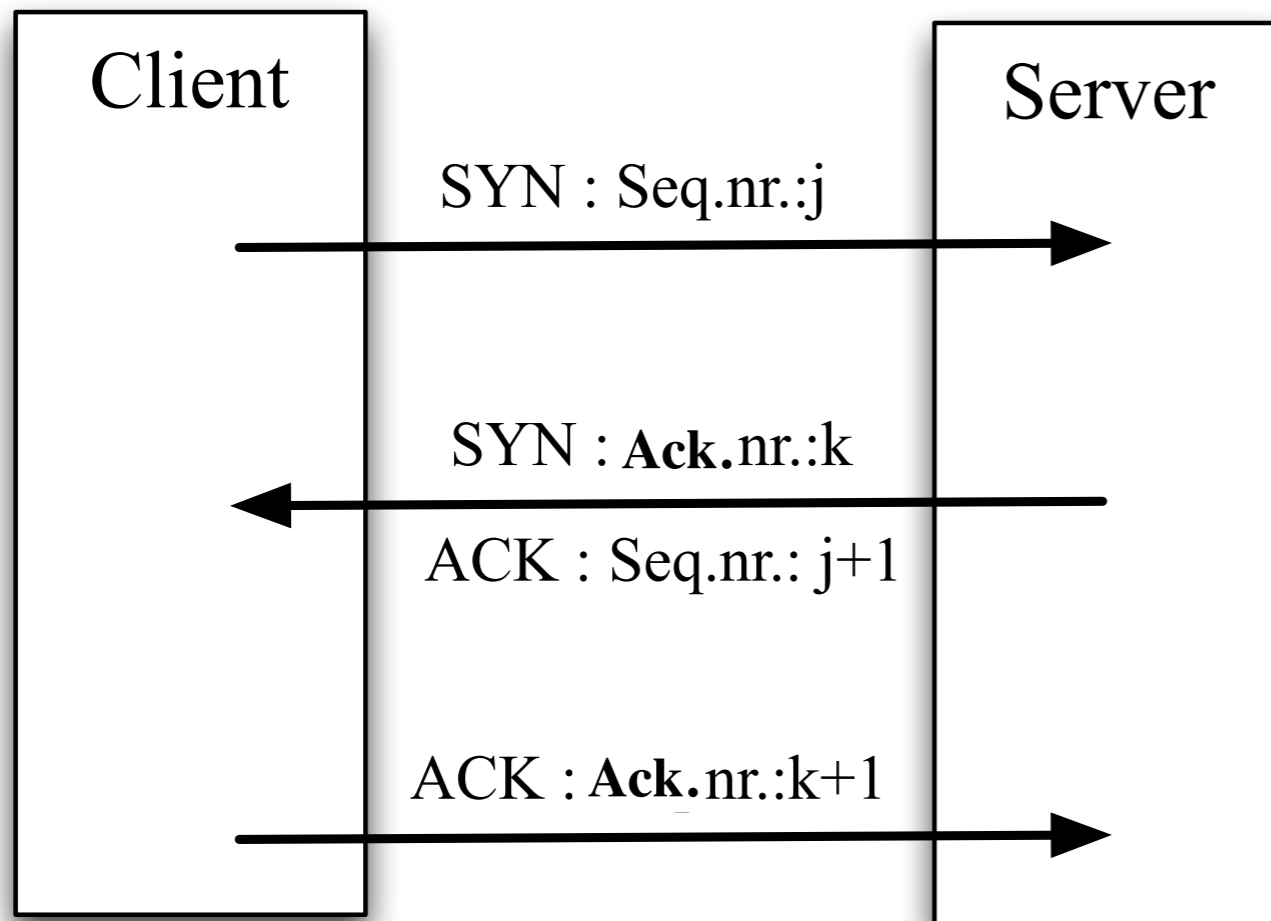
- TCP (transmission control protocol)
 - Erzeugt zuverlässigen Datenfluß zwischen zwei Rechnern
 - Unterteilt Datenströme aus Anwendungsschicht in Pakete
 - Gegenseite schickt Empfangsbestätigungen (Acknowledgments)
- UDP (user datagram protocol)
 - Einfacher unzuverlässiger Dienst zum Versand von einzelnen Päckchen
 - Wandelt Eingabe in ein Datagramm um
 - Anwendungsschicht bestimmt Paketgröße
- Versand durch Vermittlungsschicht
- Kein Routing: End-to-End-Protokolle

- TCP ist ein verbindungsorientierter, zuverlässiger Dienst für bidirektionale Byteströme
- TCP ist verbindungsorientiert
 - Zwei Parteien identifiziert durch Socket: IP-Adresse und Port
(TCP-Verbindung eindeutig identifiziert durch Socketpaar)
 - Kein Broadcast oder Multicast
 - Verbindungsaufbau und Ende notwendig
 - Solange Verbindung nicht (ordentlich) beendet, ist Verbindung noch aktiv

- TCP ist ein verbindungsorientierter, zuverlässiger Dienst für bidirektionale Byteströme
- TCP ist zuverlässig
 - Jedes Datenpaket wird bestätigt (acknowledgment)
 - Erneutes Senden von unbestätigten Datenpakete
 - Checksum für TCP-Header und Daten
 - TCP nummeriert Pakete und sortiert beim Empfänger
 - Löscht duplizierte Pakete

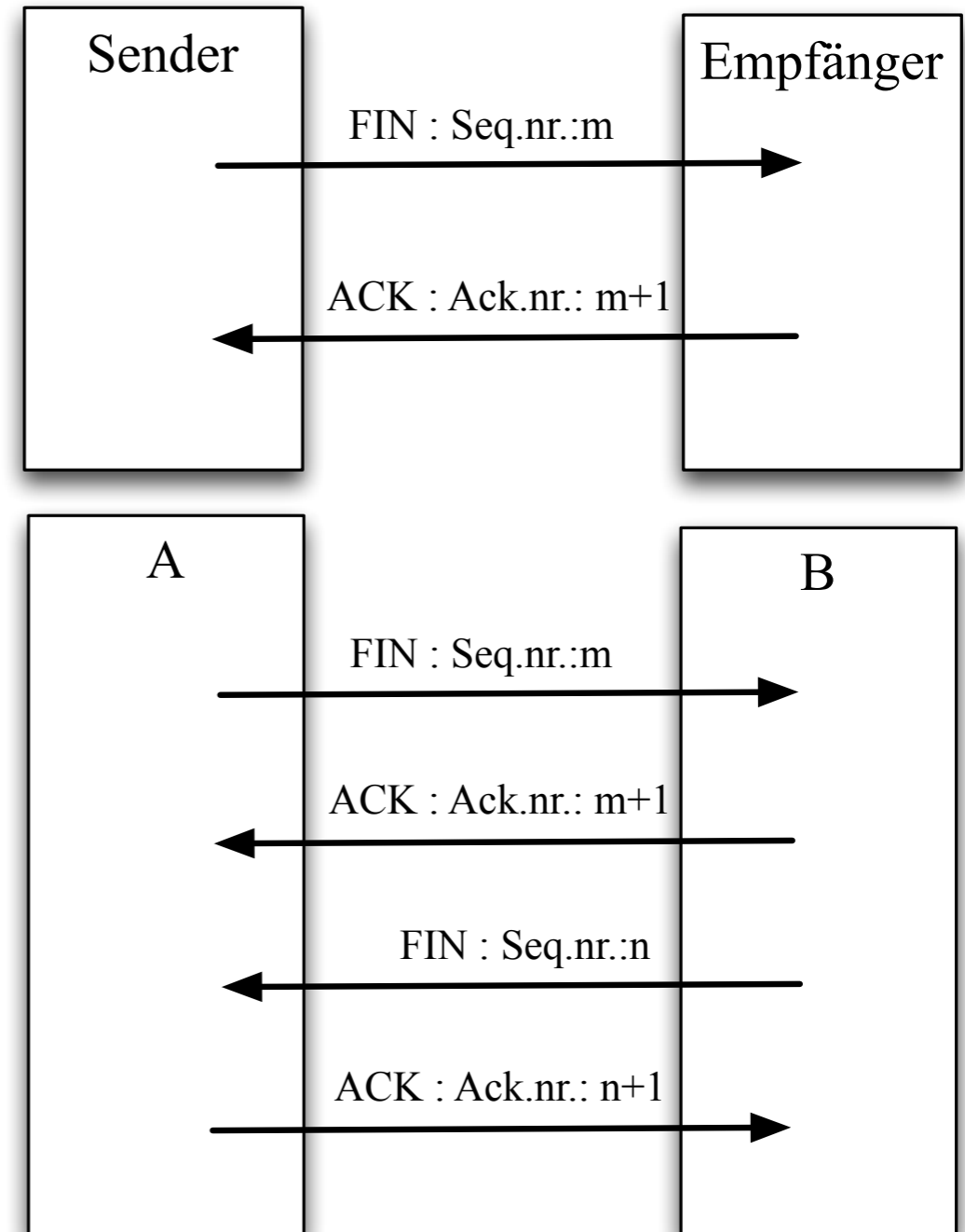
- TCP ist ein verbindungsorientierter, zuverlässiger Dienst für bidirektionale Byteströme
- TCP ist ein Dienst für bidirektionale Byteströme
 - Daten sind zwei gegenläufige Folgen aus einzelnen Bytes (=8 Bits)
 - Inhalt wird nicht interpretiert
 - Zeitverhalten der Datenfolgen kann verändert werden
 - Versucht zeitnahe Auslieferung jedes einzelnen Datenbytes
 - Versucht Übertragungsmedium effizient zu nutzen
 - = wenig Pakete

- In der Regel Client-Server-Verbindungen
 - Dann Aufbau mit drei TCP-Pakete (=Segmente)
 - Mit ersten SYN-Segment auch Übermittlung der **MSS (maximum segment size)**

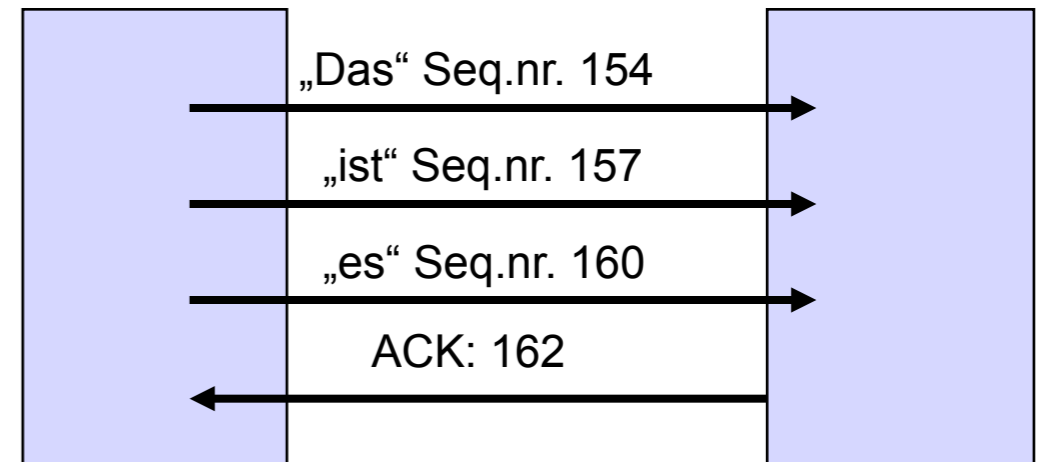
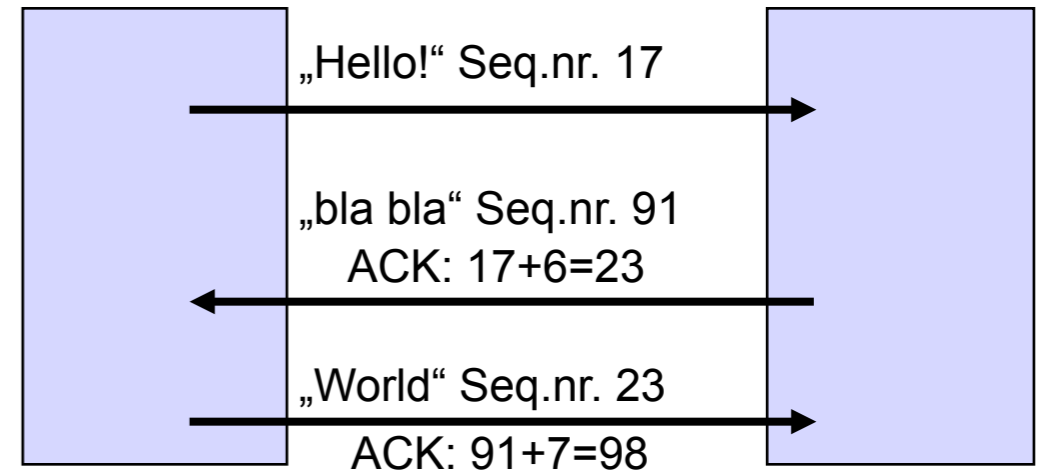


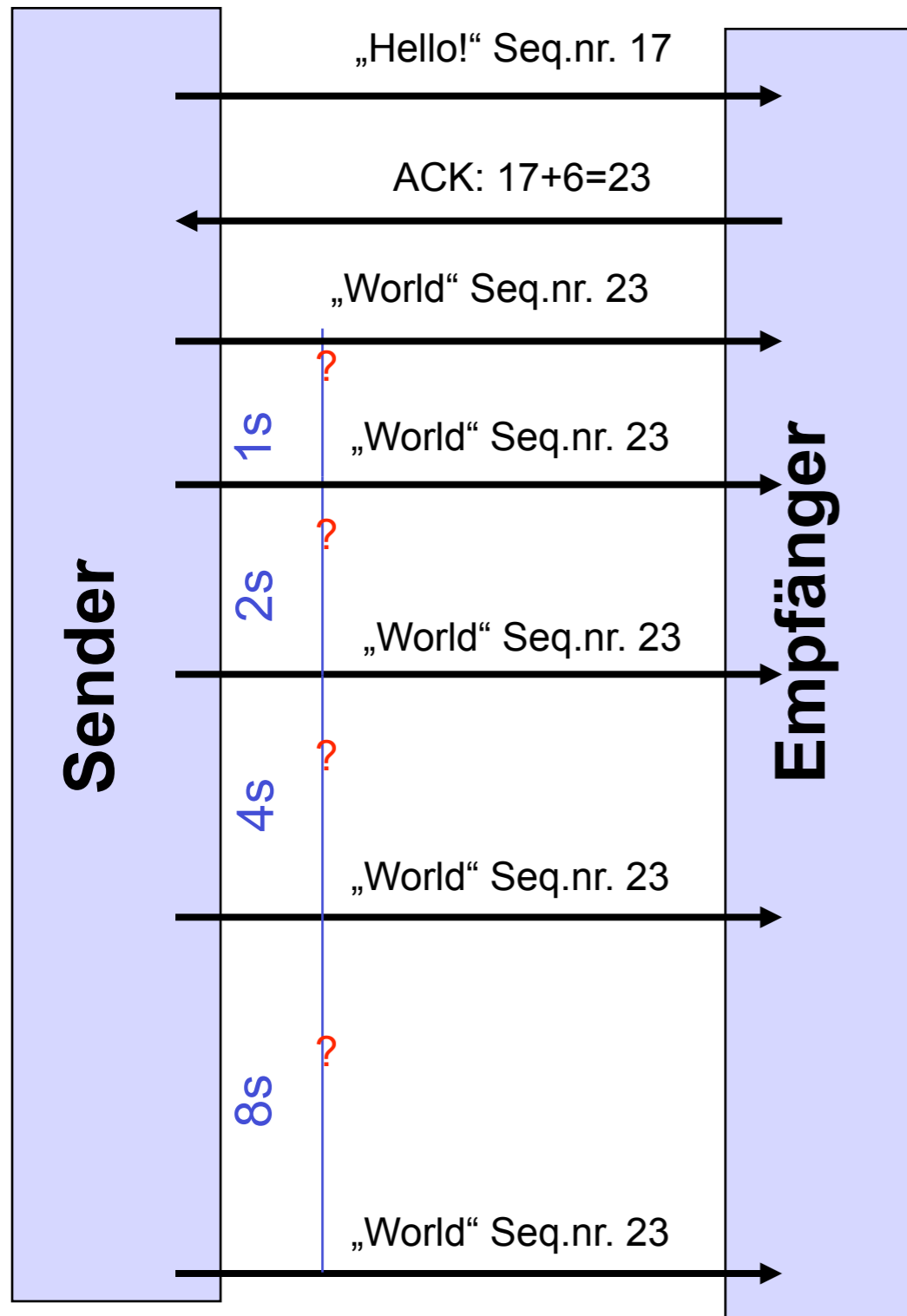
MSS (maximum segment size)

- Half-Close
 - Sender kündigt Ende mit FIN-Segment an und wartet auf Bestätigung
 - In Gegenrichtung kann weitergesendet werden
- 2 Half-Close beenden TCP-Verbindung



- Huckepack-Technik
 - Bestätigungen „reiten“ auf den Datenpaket der Gegenrichtung
- Eine Bestätigungssegment kann viele Segmente bestätigen
 - Liegen keine Daten an, werden Acks gesammelt





- **Retransmission Timeout (RTO)**
 - regelt Zeitraum zwischen Senden von Datenduplikaten, falls Bestätigung ausbleibt
- Wann wird ein TCP-Paket nicht bestätigt?
 - Wenn die Bestätigung wesentlich länger benötigt, als die durchschnittliche Umlaufzeit (**RTT/round trip time**)
 - 1. Problem: Messung der RTT
 - 2. Problem: Bestätigung kommt, nur spät
 - Sender
 - Wartet Zeitraum gemäß RTO
 - Sendet Paket nochmal und setzt
 - $RTO \leftarrow 2 RTO$ (bis $RTO = 64$ Sek.)
- Neuberechnung von RTO, wenn Pakete bestätigt werden

Schätzung der Umlaufzeit (RTT/Round Trip Time)

- TCP-Paket gilt als nicht bestätigt, wenn Bestätigung „wesentlich“ länger dauert als RTT, aber:
 - RTT nicht on-line berechenbar (nur rückblickend)
 - RTT schwankt stark
- Daher: Retransmission Timeout Value aus großzügiger Schätzung:
 - RFC 793 (1981): ($M :=$ letzte gemessene RTT)
 - $R \leftarrow \alpha R + (1 - \alpha) M$, wobei $\alpha = 0,9$ (Exponentielle Glättung)
 - $RTO \leftarrow \beta R$, wobei $\beta = 2$
 - Jacobson (1988): Schätzung nicht robust genug, daher
 - $A \leftarrow A + g (M - A)$, wobei $g = 1/8$
 - $D \leftarrow D + h (|M - A| - D)$, wobei $h = 1/4$
 - $RTO \leftarrow A + 4D$
- Aktualisierung nicht bei mehrfach versandten Pakete

- Wie kann man sicherstellen,
 - dass kleine Pakete zeitnah ausgeliefert werden
 - und bei vielen Daten große Pakete bevorzugt werden?
- Algorithmus von Nagle:
 - Kleine Pakete werden nicht versendet, solange Bestätigungen noch ausstehen.
 - Paket ist klein, wenn Datenlänge $< \text{MSS}$
 - Trifft die Bestätigung des zuvor gesendeten Pakets ein, so wird das nächste verschickt.
- Beispiel:
 - Telnet versus ftp
- Eigenschaften
 - Selbst-taktend: Schnelle Verbindung = viele kleine Pakete

- Problem: Schneller Sender, langsamer Empfänger
 - Der Sender lässt den Empfangspuffer des Empfängers überlaufen
 - Übertragungsbandweite wird durch sinnlosen Mehrfachversand (nach Fehlerkontrolle) verschwendet



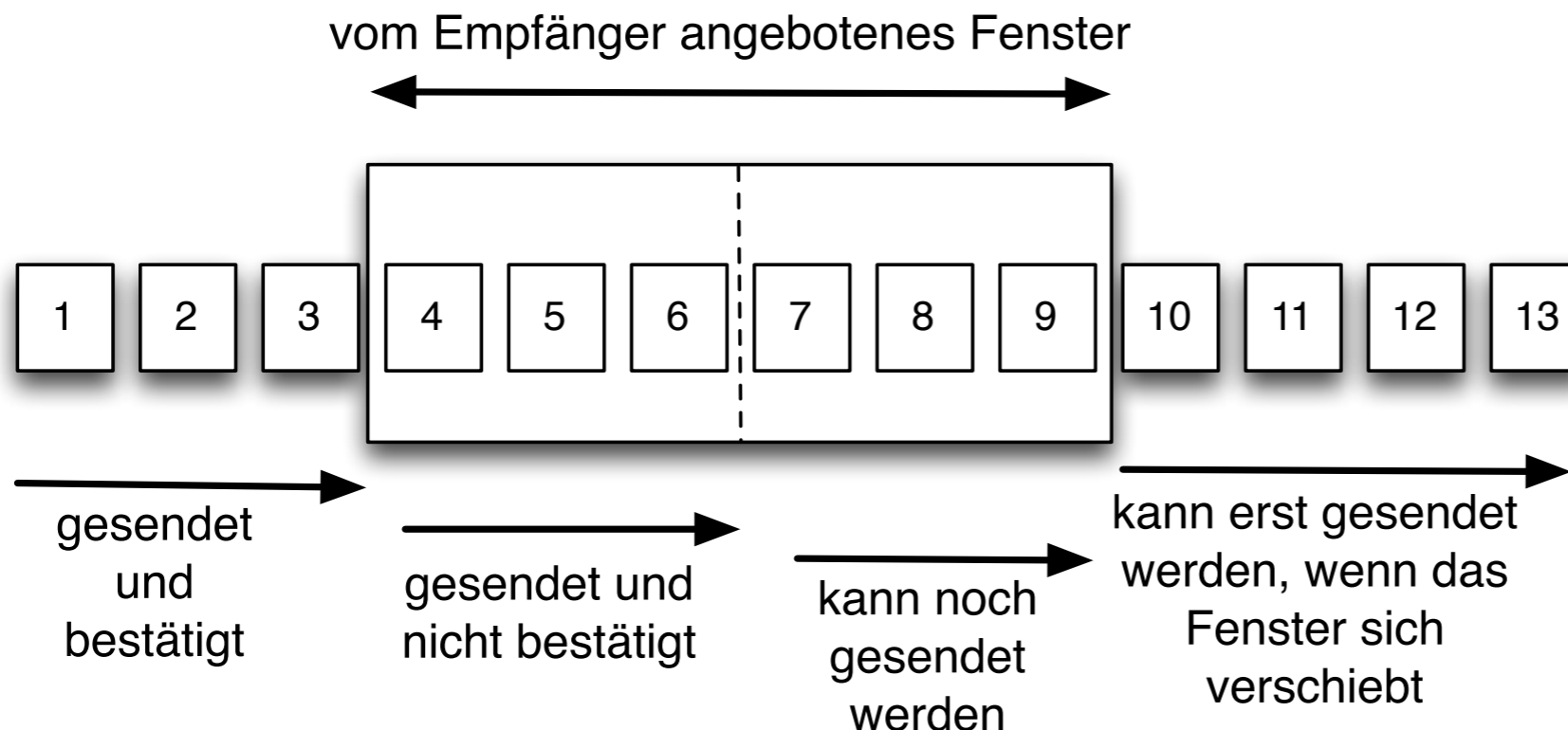
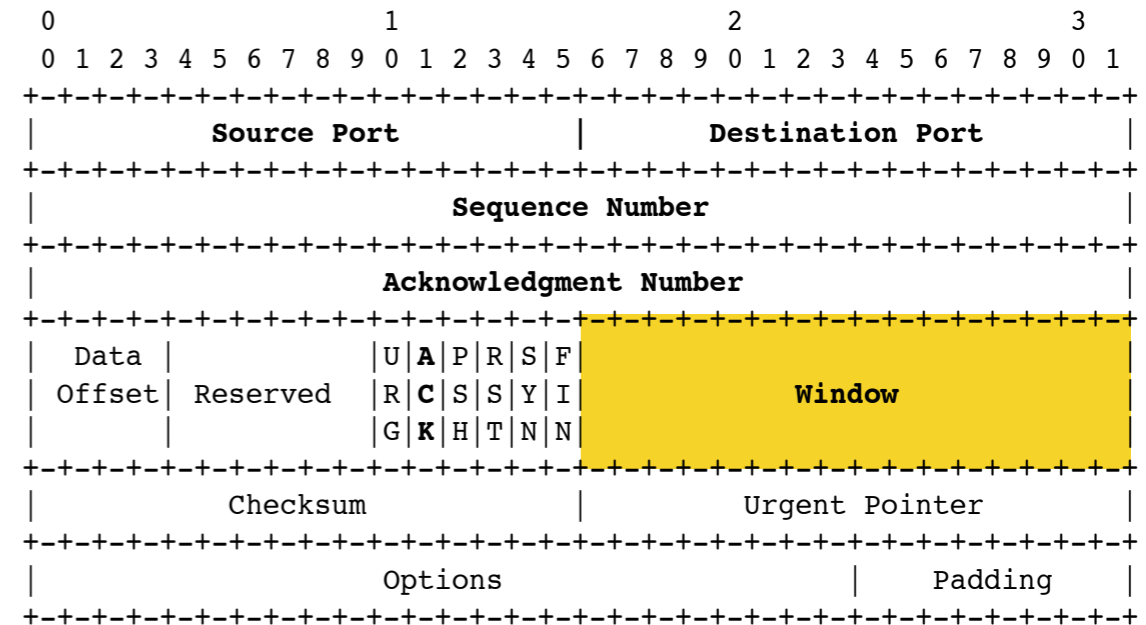
- Anpassung der Frame-Sende-Rate an dem Empfänger notwendig

■ Datenratenanpassung durch Fenster

- Empfänger bestimmt Fenstergröße (wnd) im TCP-Header der ACK-Segmente
- Ist Empfangspuffer des Empfängers voll, sendet er $wnd=0$
- Andernfalls sendet Empfänger $wnd>0$

■ Sender beachtet:

- Anzahl unbestätigter gesender Daten \leq Fenstergröße



Slow Start

Video Streaming von B nach A

A



B



- Sender darf vom Empfänger angebotene Fenstergröße nicht von Anfang wahrnehmen
- 2. Fenster: Congestion-Fenster (cwnd/Congestion window)
 - Von Sender gewählt (FSK)
 - Sendefenster: $\min \{wnd, cwnd\}$
 - S: Segmentgröße (MSS)
 - Am Anfang:
 - $cwnd \leftarrow S$
 - **Für jedes bestätigte Segment:**
 - $cwnd \leftarrow cwnd + S$
 - Solange bis einmal Bestätigung ausbleibt
- „Slow Start“ = Exponentielles Wachstum

