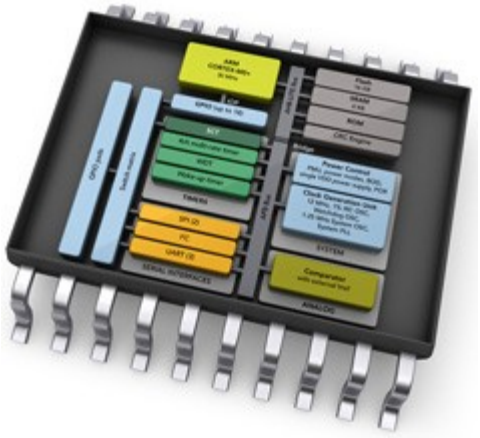
FH AACHEN
UNIVERSITY OF APPLIED SCIENCES

Kommunikationssysteme

(Modulcode 941306)

Prof. Dr. Andreas Terstegge

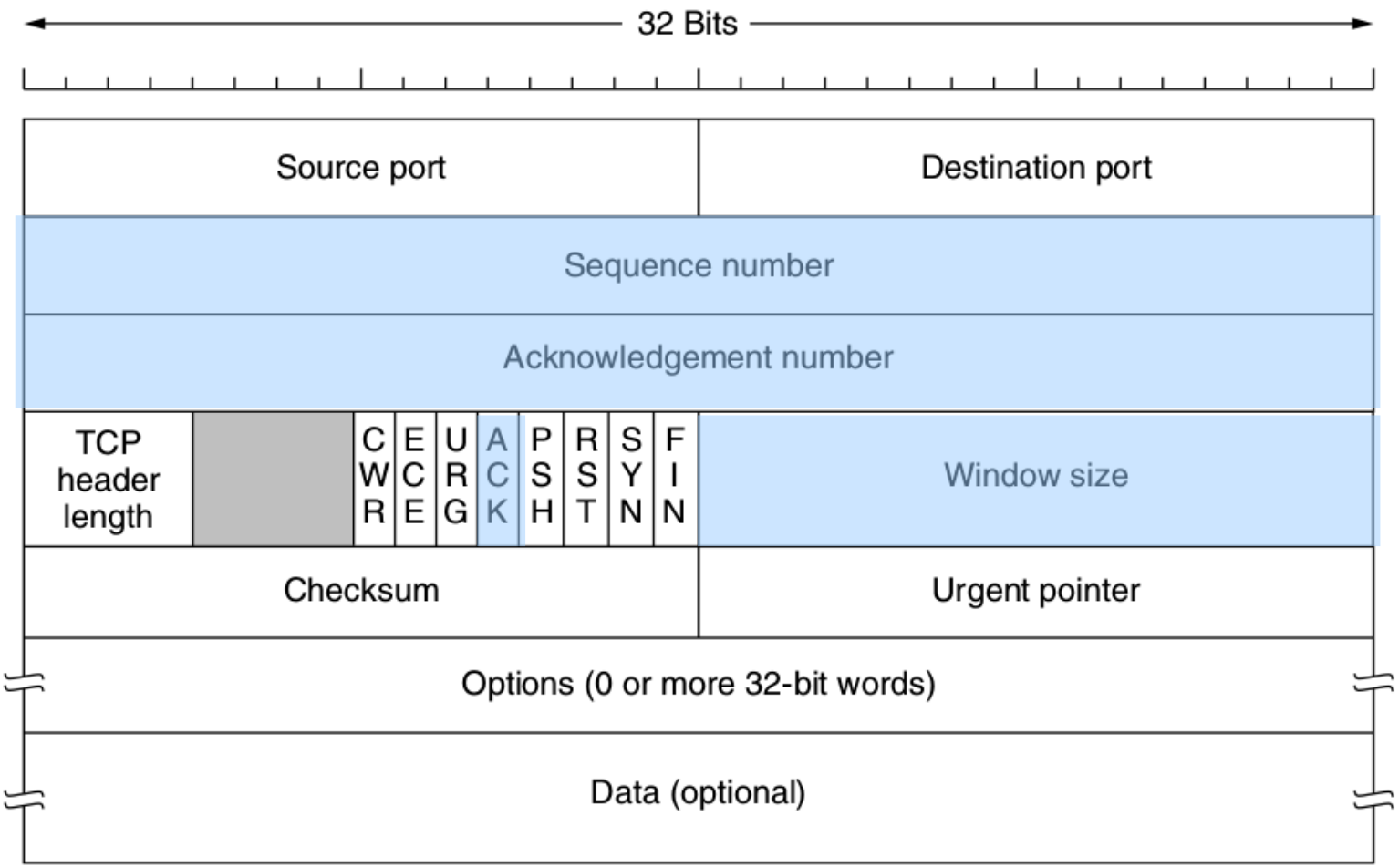


TCP: Ein Transportprotokoll mit einem dynamischen ‚Sliding-Window‘

Sliding-Window Protokoll als Grundlage

- Übertragene Segmente und ACKs müssen sich identifizieren lassen!
- Effiziente Übertragung kann nur erfolgen, wenn die Fenstergröße auf Sender- und Empfängerseite >1 ist
- Sliding-Window Protokolle können nur sinnvoll eingesetzt werden, wenn Sender und Empfänger ihre Fensterbreiten abstimmen: Situationen wie beim Go-Back-N sind zu vermeiden → es werden viele Daten auf Empfängerseite verworfen
- Fensterbreiten sind keine statischen Größen:
  Das ‚Fenster' auf Empfängerseite wird z.B. vom 'Abholen' der Daten durch die Anwendung beeinflusst!
- Mit der Fenstergröße des Senders kann die Bandbreite gesteuert werden → Flusskontrolle

TCP Header


32 Bits
Source port
Destination port
Sequence number
Acknowledgement number
TCP header length
CWR
ECE
URG
ACK
PSH
RST
SYN
FIN
Window size
Checksum
Urgent pointer
Options (0 or more 32-bit words)
Data (optional)

TCP Header

- **Sequence Number**: Byte-Offset im Datenstrom
- **Acknowledgement Number**: Nächstes erwartetes Byte (Offset)
  → kumulative Bestätigung
- **CWR** : Congestion Window reduced
- **ECE** : Explicit Congestion Notification
- **URG** : Urgent Pointer in use → Event-Erzeugung auf Gegenseite
- **ACK** : Acknowledgement Number gültig
- **PSH** : PUSH → Daten auf Empfängerseite zustellen (kein Puffern)
- **RST** : RESET
- **SYN** : Aufbau der Verbindung
- **FIN** : Abbau der Verbindung
- **Window Size** : Größe des möglichen Fensters (incl. 0)
- **Urgent Pointer** : Offset zu ‚urgent'-Daten

Flusskontrolle in TCP

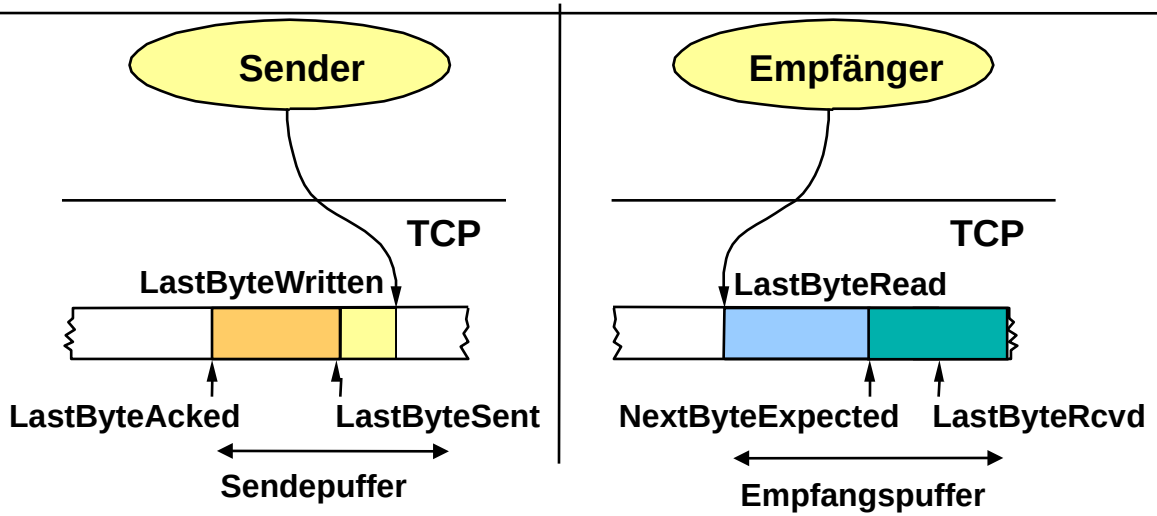
- TCP implementiert das **Sliding-Window-Protokoll**
  - Bei einer Fenstergröße von *n* können *n* Bytes verschickt werden, ohne dass ein ACK empfangen werden muss
  - Wenn der Empfang der Daten vom Empfänger bestätigt wurde, so verschiebt sich das Fenster
- Nummerierung (zyklisch mit 32-Bit)
  - Segmente werden durch ihren Byte-Offset im Stream identifiziert (Sequence Number), wobei die Startposition beim Verbindungsaufbau zufällig festgelegt wird
  - TCP verwendet kumulative ACKs: ACK n+1 sagt aus, dass alle Daten von der vorigen logischen Position bis zur Position n korrekt empfangen wurden und nun das Segment n+1 erwartet wird

Besonderheiten

- Der Empfänger puffert außerhalb der Reihenfolge empfangene Segmente, so dass die Möglichkeit besteht, Selective Repeat/Reject durchzuführen
- TCP verwendet hierbei eine **variable Fenstergröße**
  - Jede Bestätigung spezifiziert den aktuell freien Platz im Empfangspuffer (Advertised Window/Receiver Window)
  - Das Sliding Window des Senders wird somit durch die noch freie Pufferkapazität beim Empfänger beeinflusst

Wieso ist das wichtig?

Flusskontrolle in TCP: Aus dem Code



- Bedingungen für den Sender

LastByteAcked ≤ LastByteSent
LastByteSent ≤ LastByteWritten

Zwischenspeichern der Daten zwischen **LastByteAcked** und **LastByteWritten**

LastByteSent – LastByteAcked < AdvertisedWindow

EffectiveWindow = **AdvertisedWindow** - (**LastByteSent**-**LastByteAcked**)

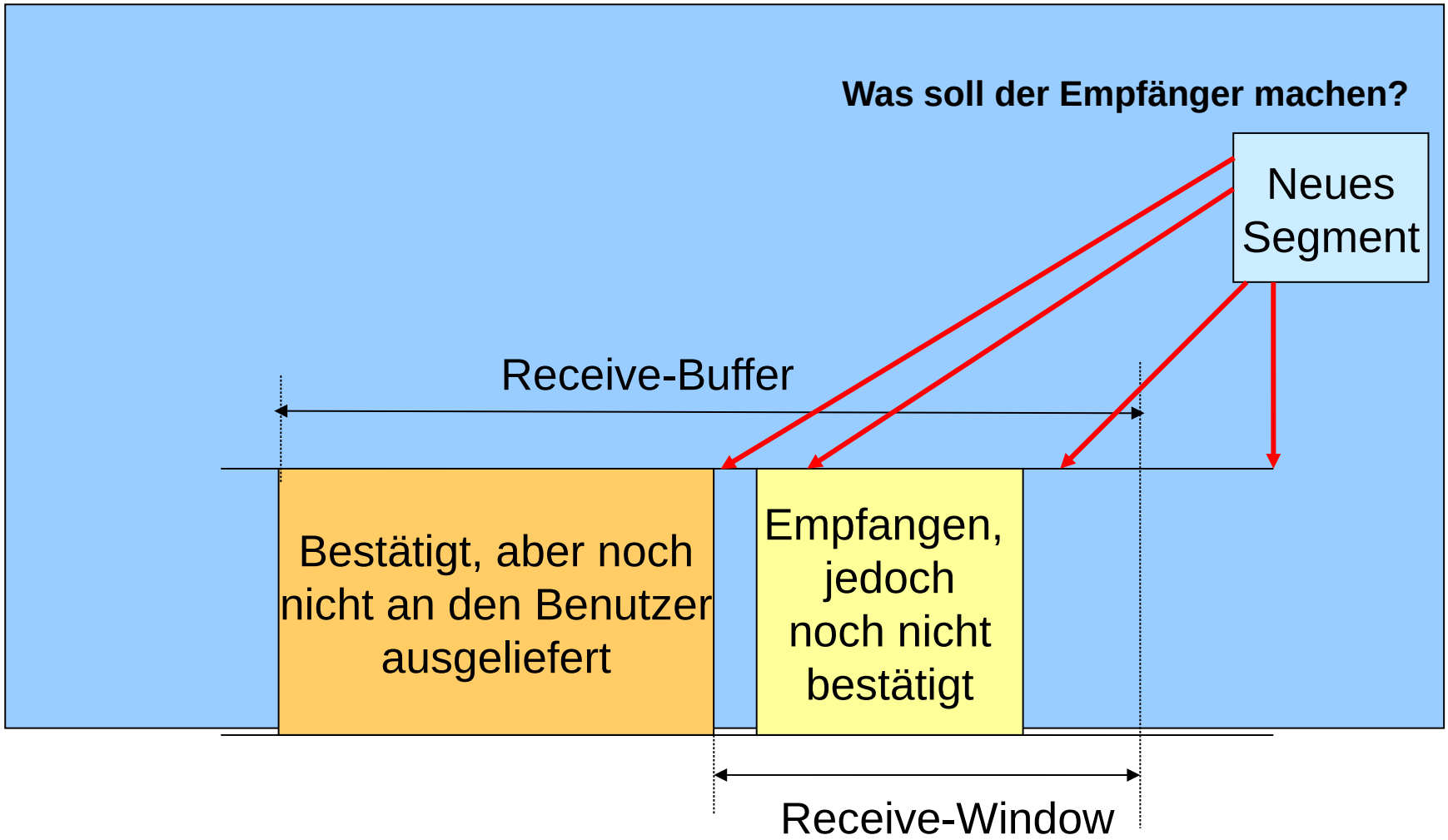
- Bedingungen für den Empfänger

LastByteRead < NextByteExpected
NextByteExpected ≤ LastByteRcvd+1

Zwischenspeichern der Daten zwischen **LastByteRead** and **LastByteRcvd**

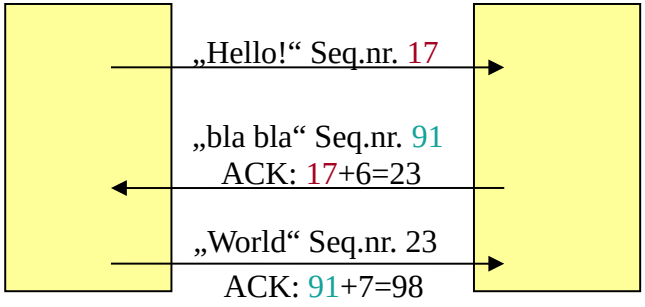
AdvertisedWindow = Empfangspuffer – ((NextByteExpected -1)-LastByteRead)

Flusskontrolle: Empfänger-Seite

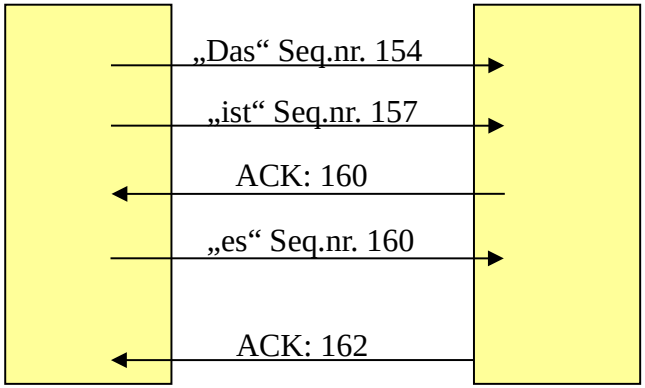


Keine eigenständigen Bestätigungen

- Huckepack-Technik
  - Bestätigungen können auf dem Datenpaket der Gegenrichtung „reiten“
- Eine Bestätigungsnachricht kann viele Segmente bestätigen
  - Kumulative Bestätigung



- Liegen keine Daten an, werden Acks verzögert – irgendwann aber doch verschickt
  - Annahme: Es kommen viele Segmente hintereinander
  - Delayed Acknowledgments
  - Nach 500ms **muss** ein ACK gesendet werden
  - Nach zwei vollständigen Segmenten **sollte** ein ACK gesendet werden



Reaktion des Empfängers (Delayed Acks)

Ereignis beim Empfänger	Aktion beim Empfänger
Ankunft eines Segments mit der erwarteten Sequenznummer. Alle Daten bis dahin sind schon bestätigt worden.	Delayed ACK. Warte bis zu 500ms auf das nächste Segment. Wenn dieses nicht empfangen wird, verschicke die Bestätigung.
Ankunft eines Segments mit der erwarteten Sequenznummer. Allerdings wurde noch keine Bestätigung des vorigen Segments verschickt.	Sofortiges Verschicken einer Kumulativen Bestätigung, die beide Segmente bestätigt.
Ankunft eines Segments hinter der erwarteten Segmentnummer. Es wird eine Lücke entdeckt.	Sofortiges Verschicken eines Duplicate ACK (DupACK). Dieses gibt die erwartete Segmentnummer an.
Ankunft eines Segments, das eine Lücke teilweise oder vollständig füllt.	Sofortiges Verschicken einer Bestätigung.

Transmission Control Protocol: Eine hybride Lösung

Go-Back-N/Selective Repeat Protokoll: TCP-Sender verwaltet lediglich einen Timer für das Segment, welches als nächstes bestätigt werden muss. Kommt es zu einem Time-Out, so wird, bedingt durch das Zwischenspeichern auf Empfängerseite, hier zu einem **Selective Repeat** durchgeführt.

Selective Reject: Empfänger speichert nicht nur out-of-Order Segmente im Empfangspuffer. Die meisten Versionen von TCP **emulieren** NAK-Mechanismen mittels dreifachem ACK mit gleicher Sequenznummer:

Hierdurch kann ein **erneutes Übertragen eines Segments VOR Timeout** initiiert werden (beim 3. Duplicate Acknowledgement) Wir haben somit eine Art **Selective Reject**. Die kumulative Bestätigungen ermöglichen das **Nutzen zwischengepufferter Daten** (Datenlücken können geschlossen werden)

Transmission Control Protocol: Timer

- **Retransmission Timer**
  Timer wird beim Senden eines Segmentes gestartet.
  Bei Ablauf: Retransmission!

- **Persistence Timer**
  Timer wird nach Empfang einer Fenstergröße von 0 gestartet. Nach Ablauf Anfrage nach neuem Fenster (>0)

- **Keepalive Timer**
  Rel. lange Wartezeit. Wir bei jeder Transaktion rückgesetzt. Nach Ablauf → Abbau der Verbindung

TCP Round Trip Time und Timeout

Q: Wann sollte es ein Timeout geben?

- RTT ist die Zeit, die eine Antwort aktuell (mindestens) braucht
- Timeout sollte nicht vor dem RTT kommen
- Problem: RTT variiert
- **Zu kurz:** vorzeitiger Timeout und damit unnötige Neuübertragungen
- **Zu groß:** langsame Reaktion auf Paketverluste → geringer Datendurchsatz

Q: Wie kann die RTT geschätzt werden?

- TCP stoppt die Zeit zwischen dem Versenden eines Segmentes und dem Empfang des ACKs
- Neuübertragungen gehen hier nicht ein, Delayed Acks müssen ausgeblendet werden
- **Smoothed RTT** wird zur Schätzung der RTT geglättet

$$SRTT = \alpha SRTT + (1-\alpha) R$$

- Hierdurch ergibt sich eine Mittelung über die Vergangenheit, und nicht nur die aktuelle **Smoothed RTT**

TCP ist eigentlich noch viel mehr…

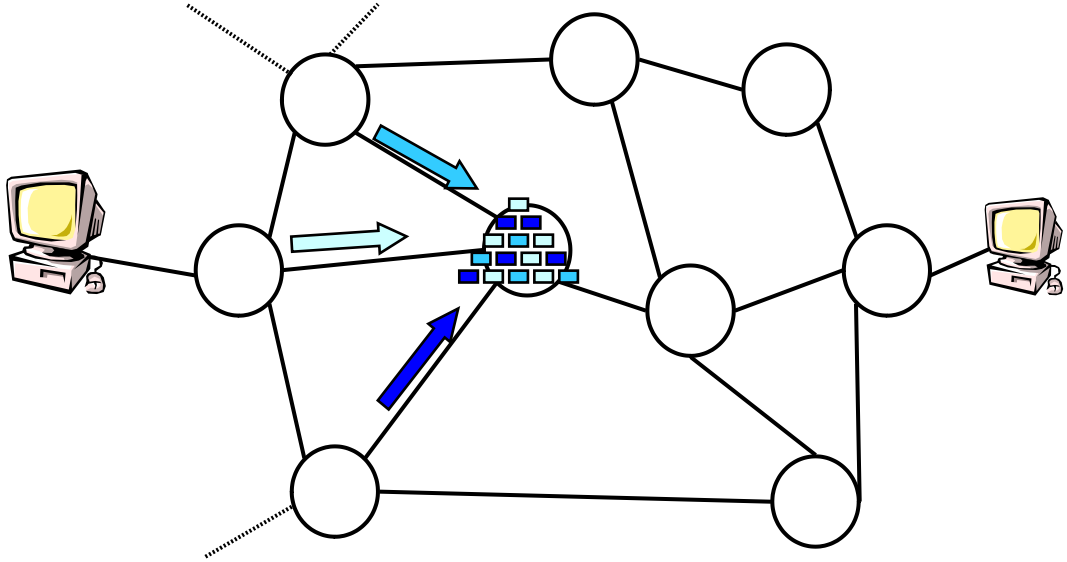
- Der Verlust von Segmenten führt ggf. zum Selective Repeat und damit zu erheblichen Neuübertragungen!
  - Einigen sich Sender und Empfänger (durch ein großes Fenster bei der Flusskontrolle) auf eine hohe Datenrate, wird das Netz ggf. stark belastet
  - Existieren viele solche Übertragungen im Netz, können Router überlastet werden. Als Resultat verwerfen sie Pakete, so dass auf TCP-Ebene keine Quittungen mehr eingehen
  - TCP wiederholt die Daten und belastet das Netz damit noch stärker

→ Die Staukontrolle bei TCP berücksichtigt auch noch den **Netzzustand**

  - Slow Start, Congestion Avoidance, Fast Retransmit, Fast Recovery, Selective ACK, …

TCP – Überlastvermeidung

- TCP ist vom zugrundeliegenden Netz getrennt
  - Großes Fenster der Flusskontrolle: hohe Datenrate, Netz stark belastet
  - Durch viele Verbindungen können Router **überlastet** werden
    - > Überlast/Verstopfung: engl.: **Congestion**
    - > Pakete werden verworfen, auf TCP-Ebene gehen keine Quittungen ein
    - > TCP wiederholt die Daten und belastet das Netz damit noch stärker



TCP: Flusskontrolle/Staukontrolle

Flusskontrolle:

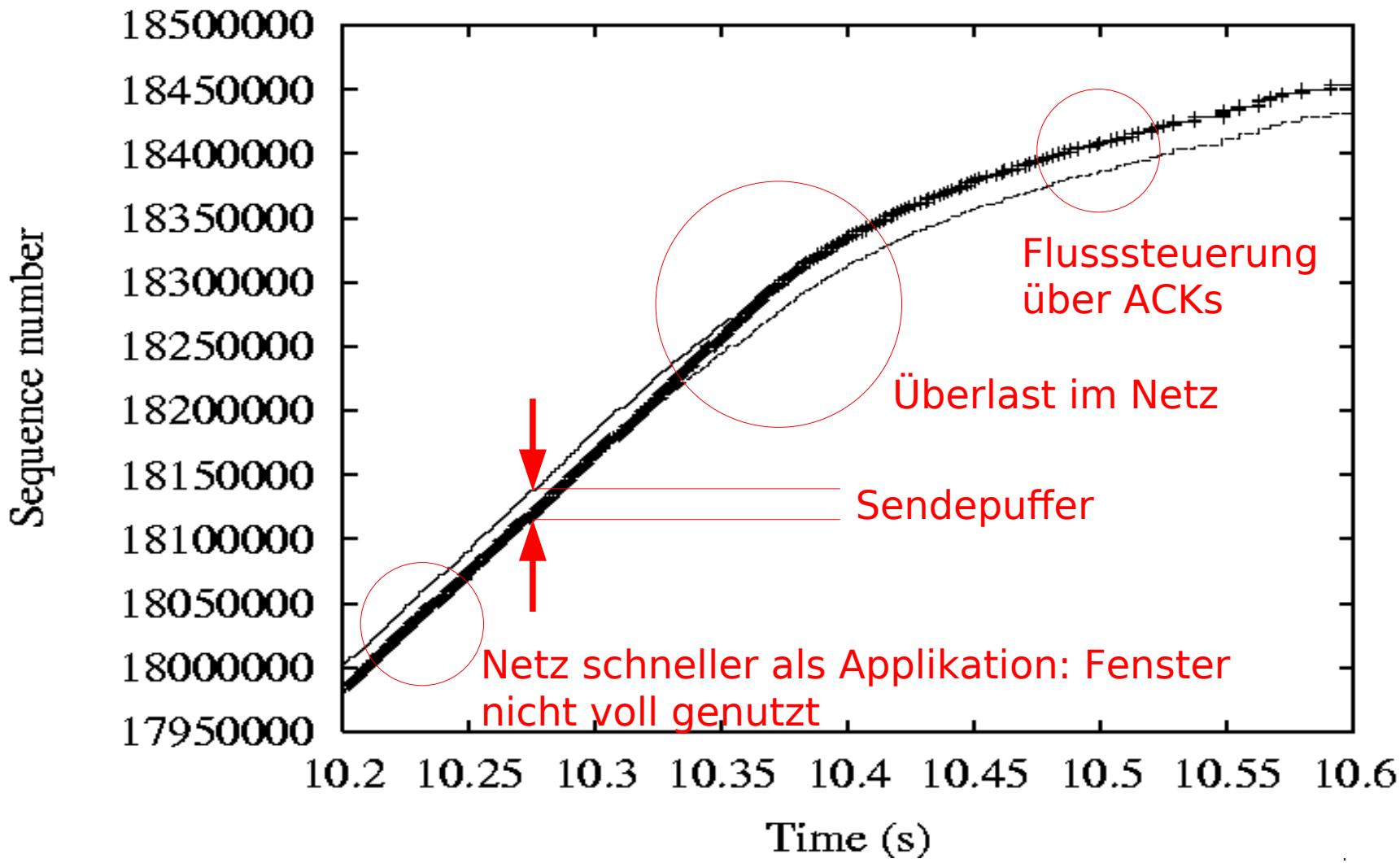
- basiert auf dem Sliding-Window-Protokoll
- regelt den Datenfluss zwischen den Transportdienst-Nutzern
- Ack-getriebenes injizieren von Daten

Staukontrolle:

- befasst sich mit Überlastsituationen in den Zwischensystemen (Routern)
- Überlastsituationen in Zwischensystemen führen zu Paketverlusten, so dass die entsprechenden Segmente nach einer gewissen Zeit erneut übertragen werden, bei Time-Out kommt es gar zu einem „Go-Back-N“/Selective Repeat

  → Verstärkung der Überlastsituation durch unnütze Übertragung (congestion collapse))!

Flusskontrolle in TCP: Self-Clocking

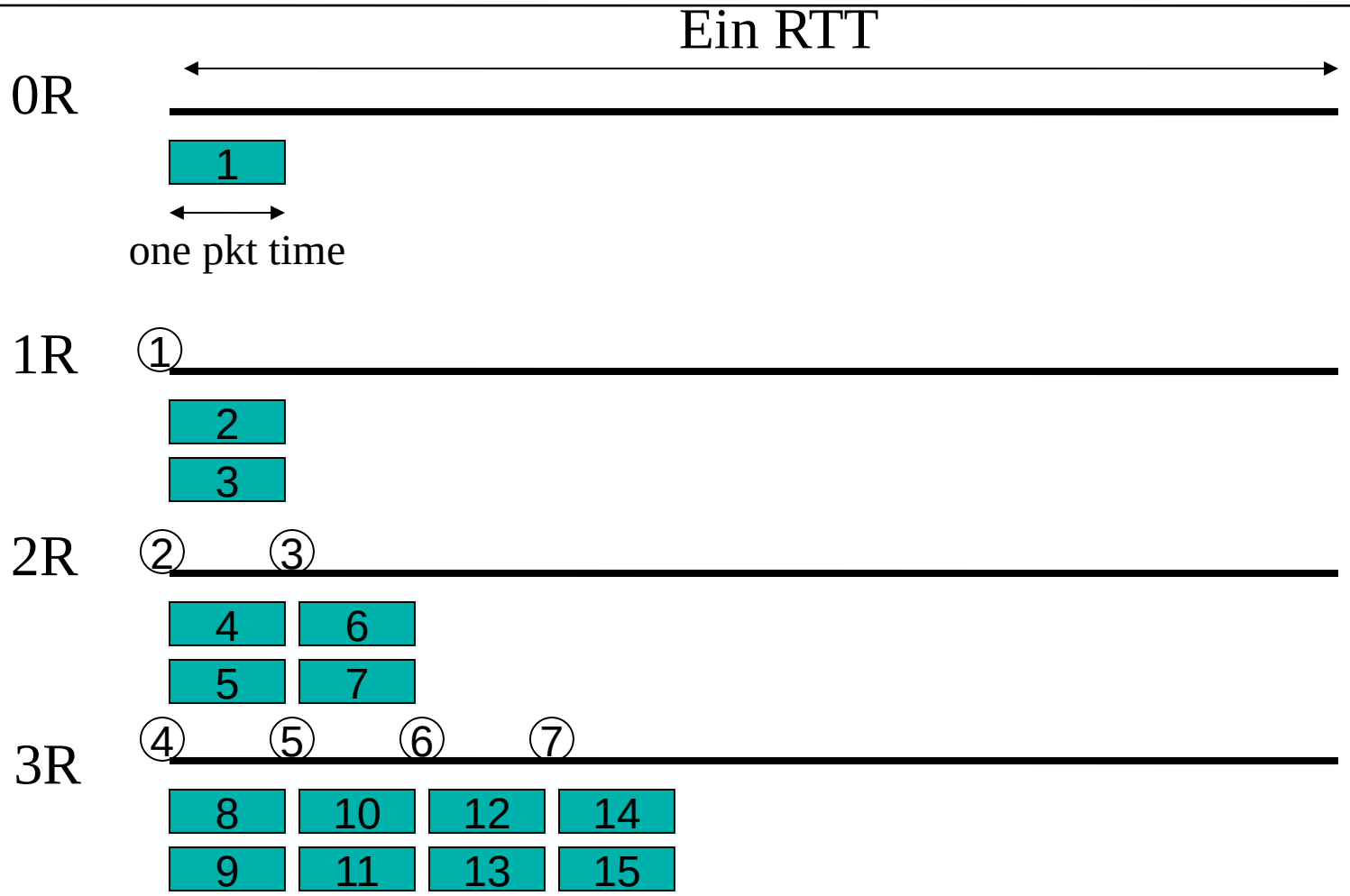


TCP: Staukontrolle

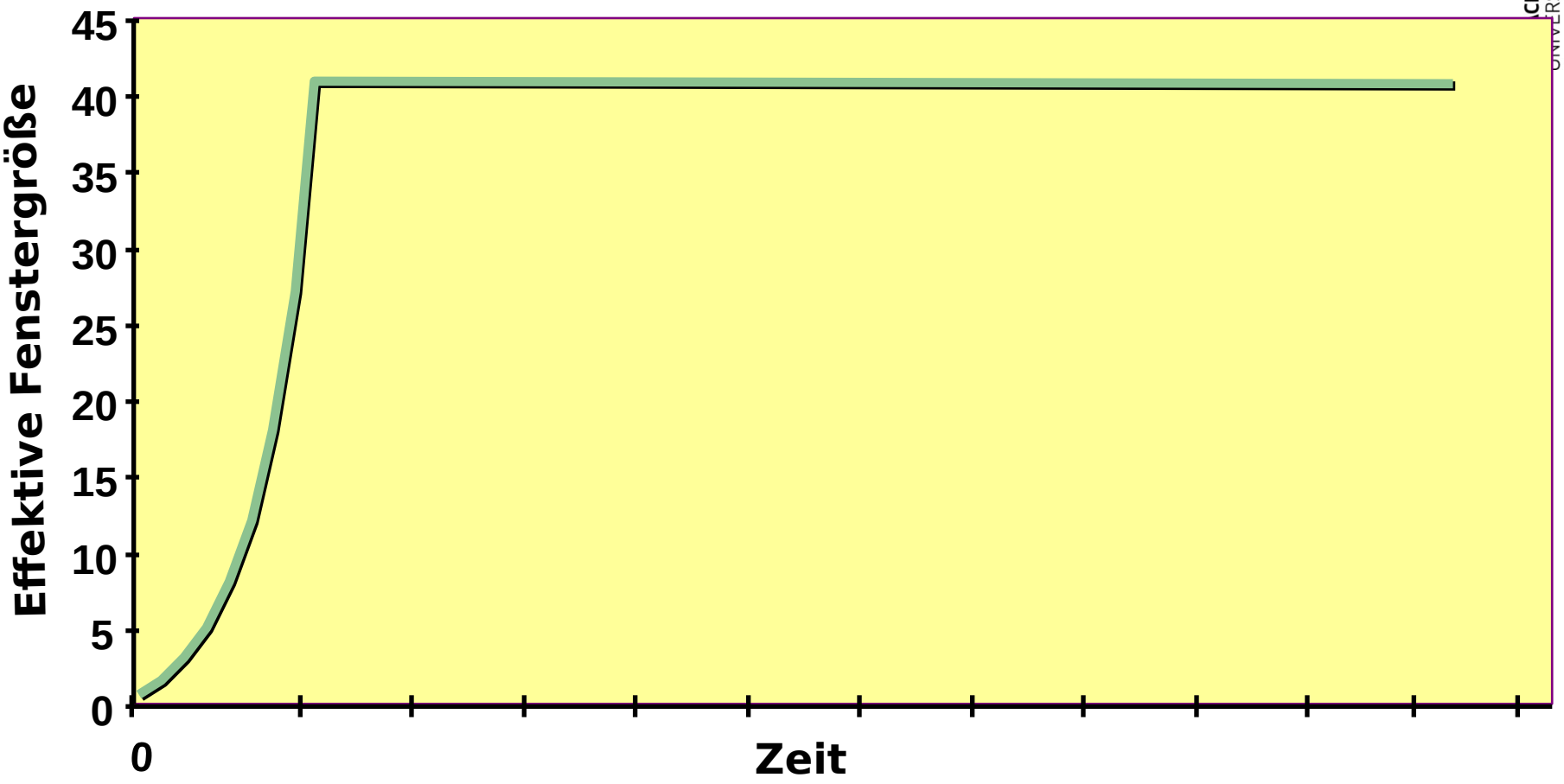
- **Überlastfenster (Congestion-Window):**
  - Zusätzliche Beschränkung des verfügbaren Fensters durch ein **Congestion Window**
  - Pflege eines internen Thresholds, der das Vorgehen beeinflusst (**s**low-**s**tart **thresh**old):

    ssthresh = min (Receiver Window, Congestion Window) [Byte]
  - Congestion Window bestimmt insbesondere zu Beginn die Übertragungsrate (Slow-Start)
  - Initiale Größe = 1-4 Segmente maximaler Größe
  - Größe des Überlastfensters wird in der Slow-Start-Phase bei jeder erfolgreichen Bestätigung um die Größe eines Segments erhöht
    > erfolgreiche Übertragung eines vollen Fensters, d.h. von *n* Segmenten (Burst), verdoppelt somit die Größe → exponentielles Wachstum
    > maximal bis Größe des Receiver-Windows erreicht

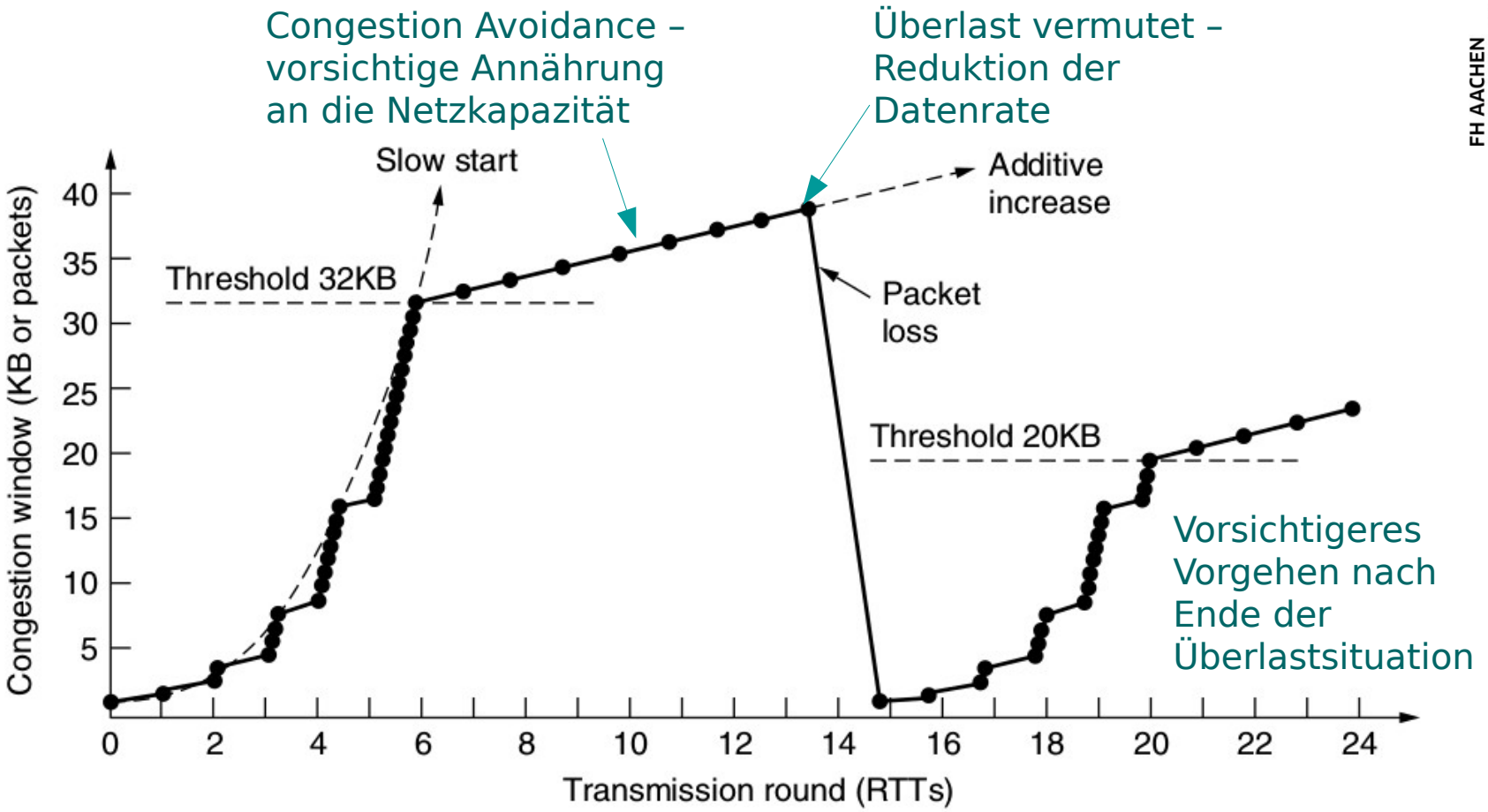
Slow-Start-Beispiel



Staukontrolle – Slow-Start

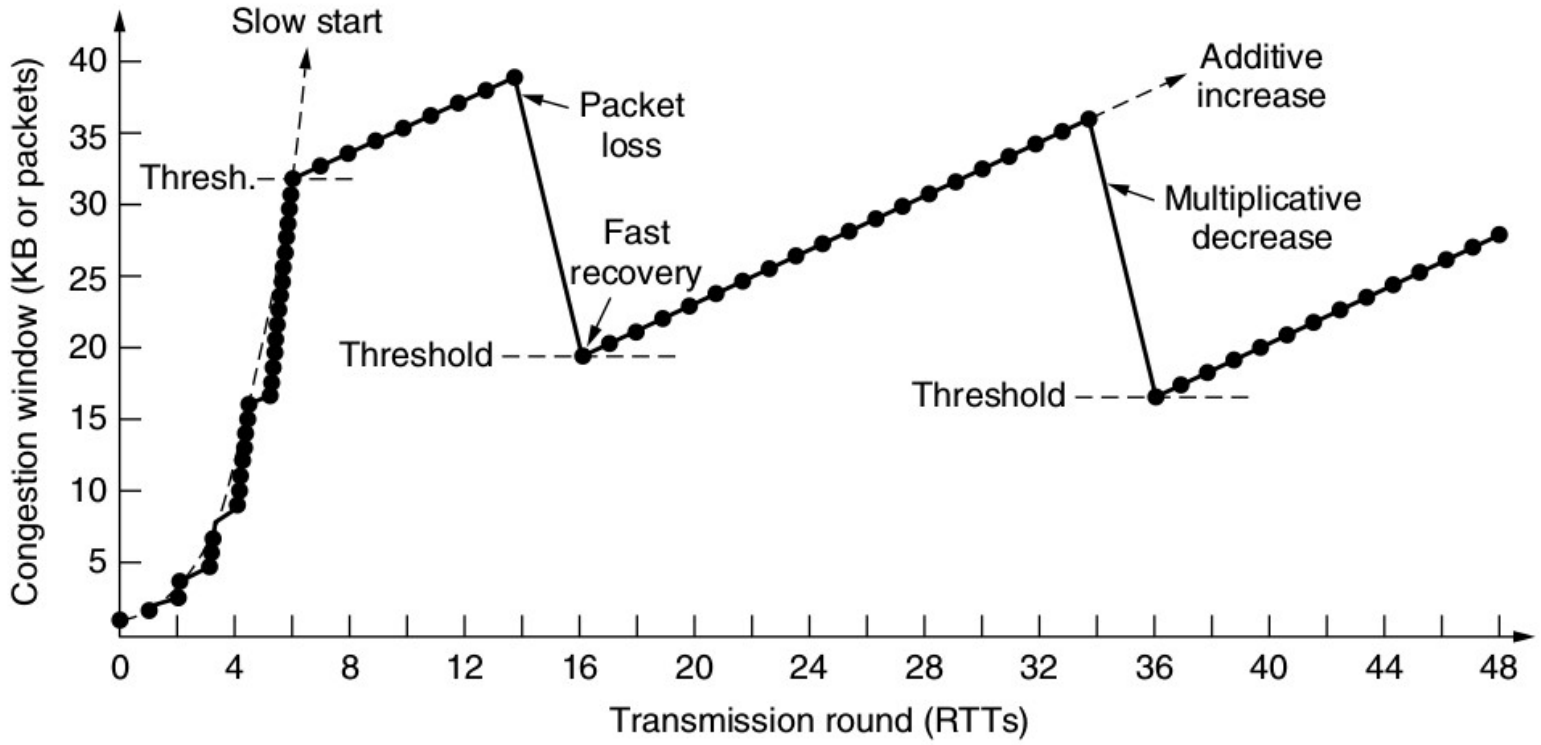


Beispiellauf von Slow Start / Congestion Avoidance (TCP Thaoe)



- Vorsichtiger Beginn (Slow-Start), aber nur bis zu einem Schwellwert
- Keine Unterscheidung zwischen Timeout und duplicate ACKs

Fast Retransmit und Fast Recovery (TCP Reno)



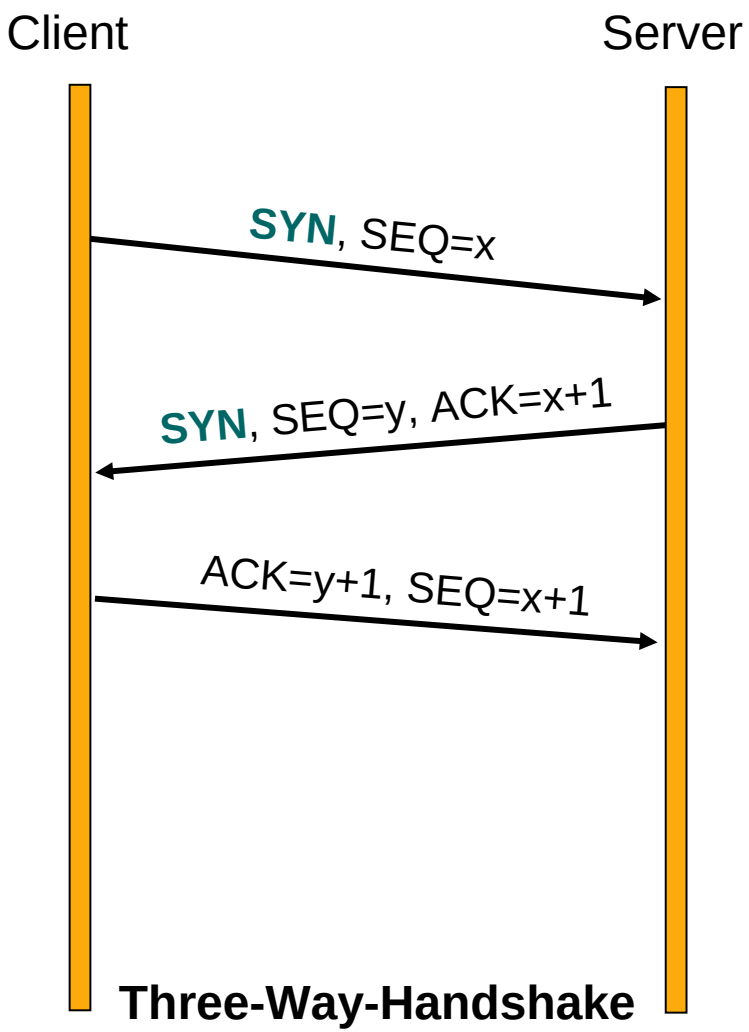
- Slow Start wie bei TCP Thaoe
- Unterscheidung zwischen Timeout und duplicate ACKs:
  - Bei Timeout gleiches Verhalten wie TCP Thaoe
  - Bei dup-ACKs (selective Reject) → Halbierung des Congestion Window (fast recovery)

Staukontrolle bei TCP

- Congestion Control ist deutlich komplexer als hier dargestellt
  - Viele Erweiterungen, um Einbruch der Datenrate zu vermeiden
  - ...

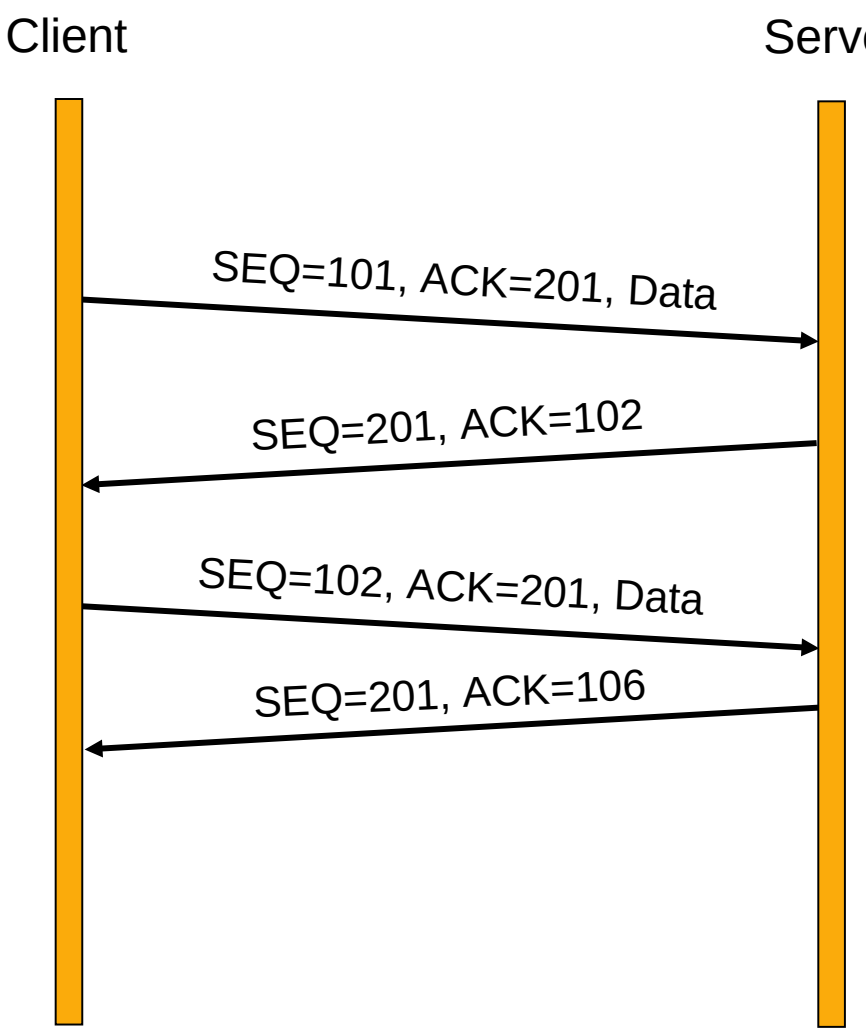
- Staukontrolle schafft Fairness! Wird eine Leitung von N-TCP-Verbindungen genutzt, so erhät jede 1/N-Anteile der verfügbaren Bandbreite

TCP-Verbindungsmanagement: 1. Verbindungsaufbau



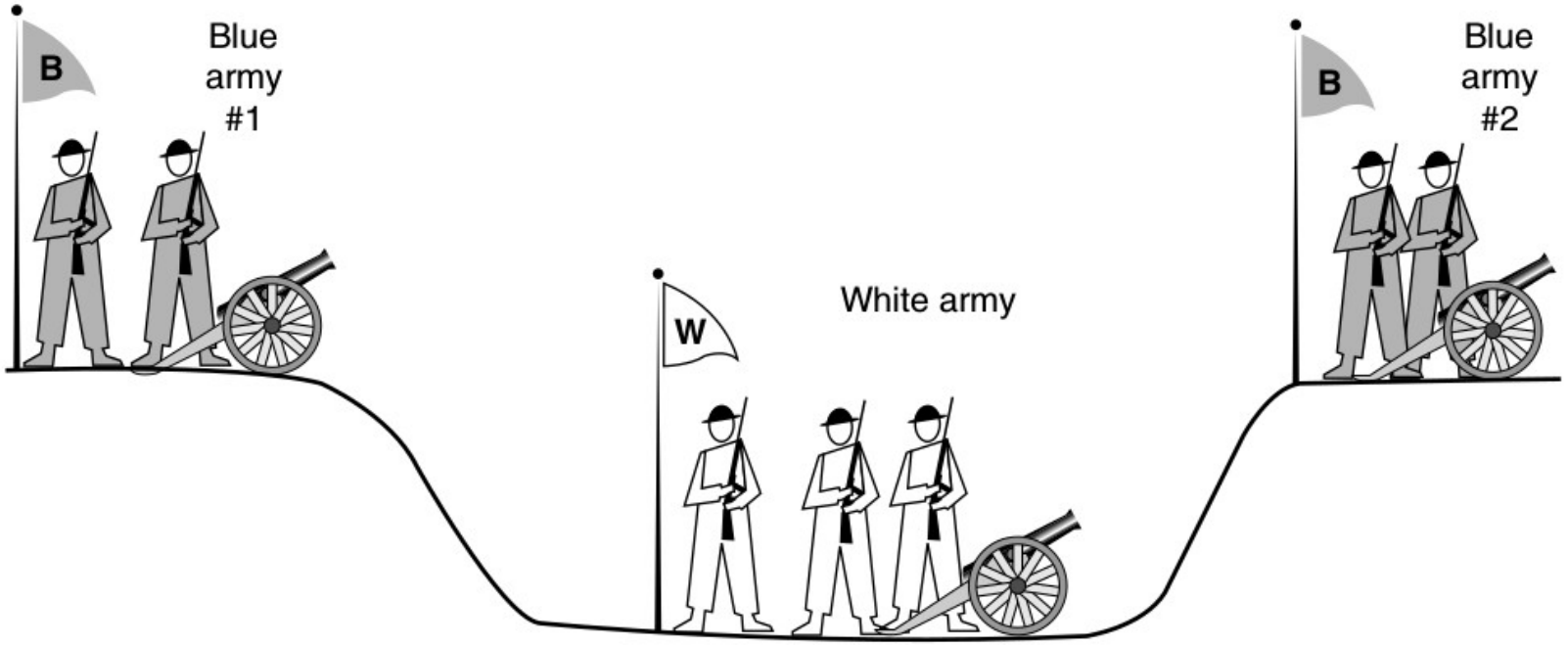
- Der Server wartet auf eingehende Verbindungswünsche.
- Der Client führt unter Angabe von IP-Adresse, Portnummer und maximal akzeptabler Segment-Größe eine CONNECT-Operation aus
- CONNECT sendet ein SYN
- Ist der Destination Port der CONNECT-Anfrage identisch zu der Port-Nummer, auf der der Server wartet, wird die Verbindung akzeptiert, andernfalls mit RST abgelehnt
- Der Server schickt seinerseits das SYN zum Client und bestätigt zugleich den Erhalt des ersten SYN-Segments
- Der Client schickt eine Bestätigung des SYN-Segments des Servers. Damit ist die Verbindung aufgebaut

TCP-Verbindungsmanagement: 2. Datenübertragung



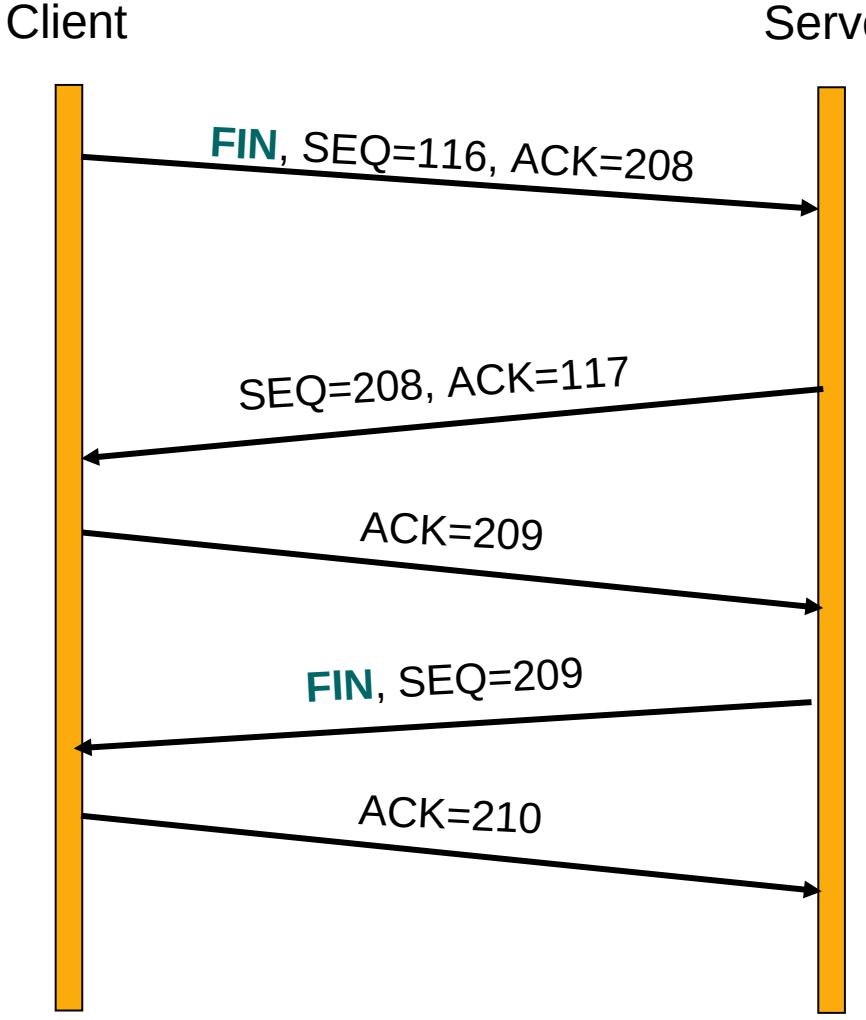
- Vollduplex-Betrieb
- Aufteilung eines Bytestroms in Segmente. Übliche Größen sind 1460, 536 oder 512 Byte; dadurch wird IP-Fragmentierung vermieden.
- Üblicher Quittungsmechanismus: Alle Segmente bis ACK-1 sind bestätigt. Hat der Sender vor dem Empfang eines ACKs einen Timeout, überträgt er erneut.

TCP-Verbindungsmanagement: 3. Verbindungsende



- Wann kann die blaue Armee angreifen? (2*Blue > White, Blue < White)
- Benachrichtigung über Boten…
- Ist das sicher?
- Analogie zum Verbindungsabbau bei TCP: Wann sind sich beide Seiten sicher, dass die Verbindung abgebaut wurde?

TCP-Verbindungsmanagement: 3. Verbindungsende



- Abbau als zwei Simplex-Verbindungen
- Senden eines FIN-Segments
- Wird das FIN-Segment bestätigt, wird die Richtung 'abgeschaltet'. Die Gegenrichtung bleibt aber noch offen, hier kann noch weiter gesendet werden.
- Verwendung von Timern zum Schutz vor Paketverlust.

Transmission Control Protocol: Timer

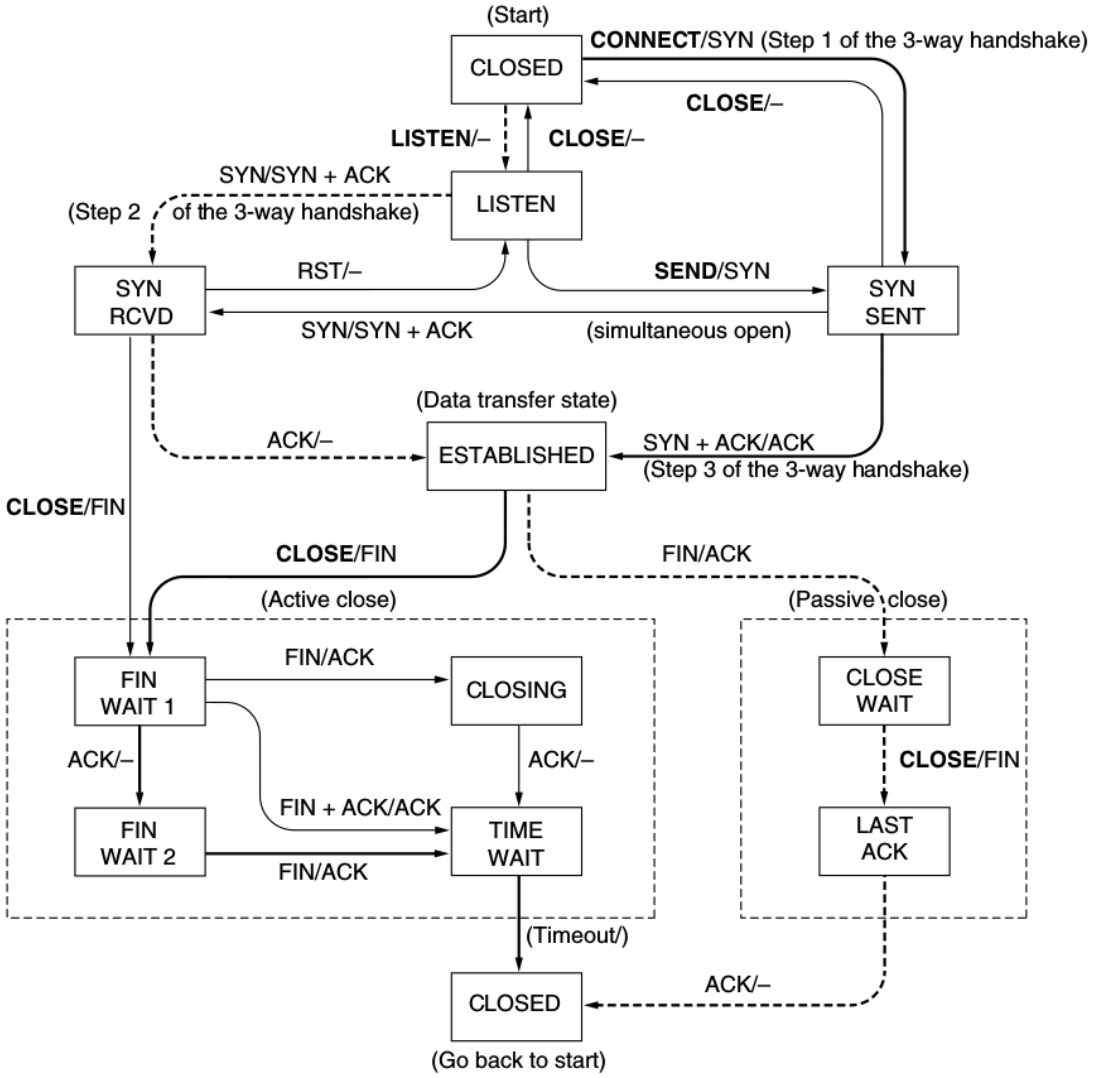
- **Retransmission Timer**
  Timer wird beim Senden eines Segmentes gestartet.
  Bei Ablauf: Retransmission!

- **Persistence Timer**
  Timer wird nach Empfang einer Fenstergröße von 0 gestartet. Nach Ablauf Anfrage nach neuem Fenster (>0)

- **Keepalive Timer**
  Rel. lange Wartezeit. Wir bei jeder Transaktion rückgesetzt.
  Nach Ablauf → Abbau der Verbindung

- **Timer in TIME_WAIT**
  Folgt auf ein FIN innerhalb von 2 RTTs kein ACK, wird die Verbindung abgebaut!

TCP-Verbindungsmanagement: Übersicht Zustandsautomat



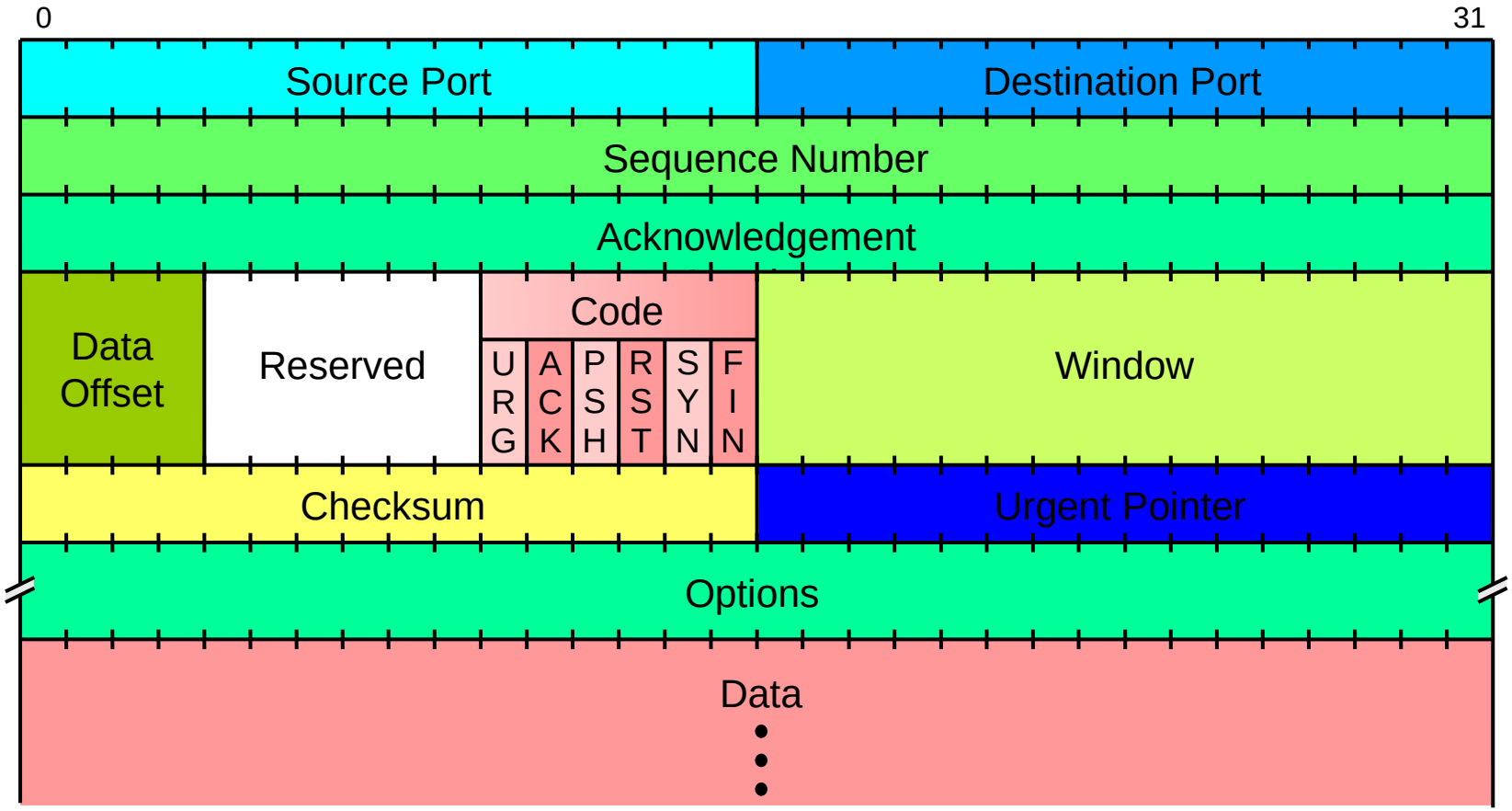
Syntax der Übergänge:
Ereignis/Aktion

Fette Linie:
Normaler Pfad des Clients

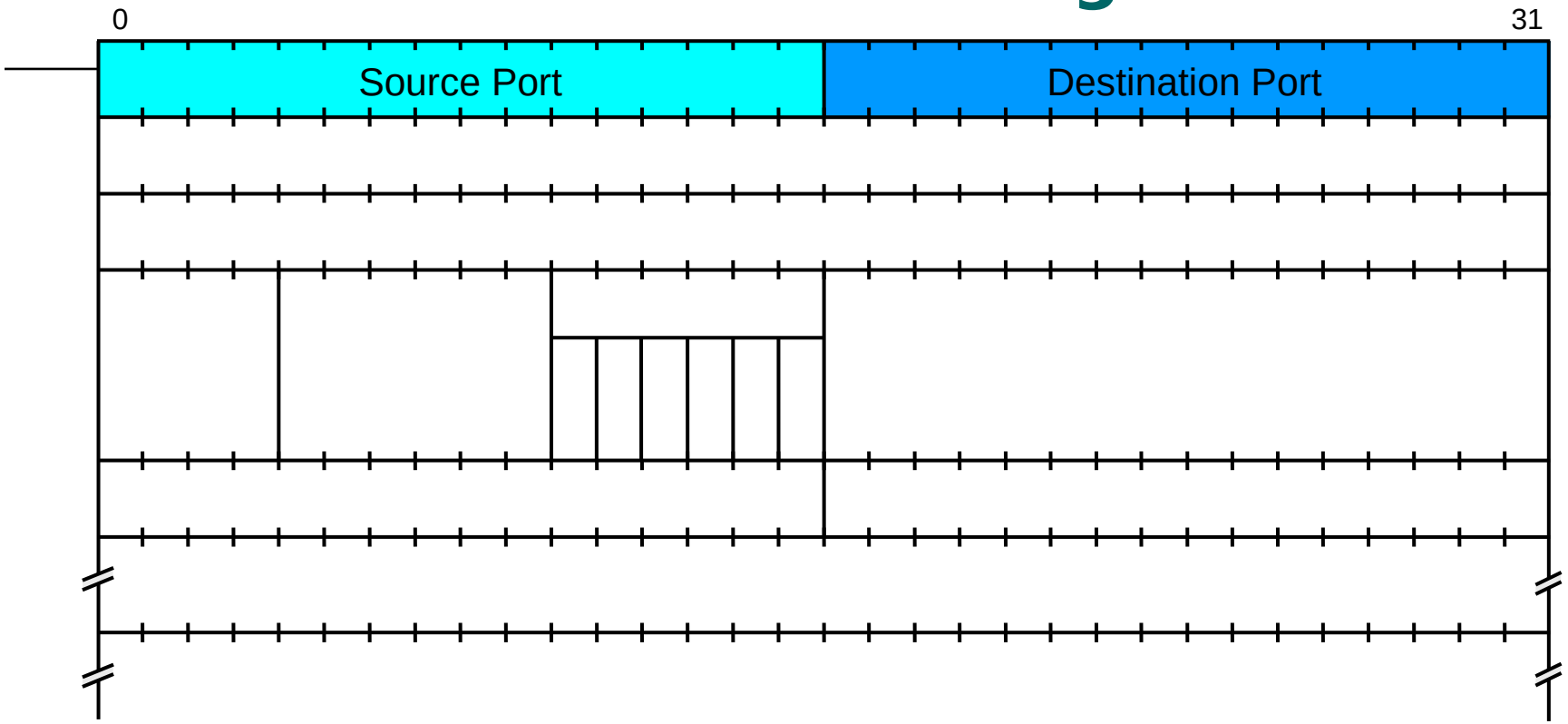
Fette gestrichelte Linie:
Normaler Pfad des Servers

Dünne Linien:
Ungewöhnliche Ereignisse

Format eines TCP-Segments



Format eines TCP-Segments



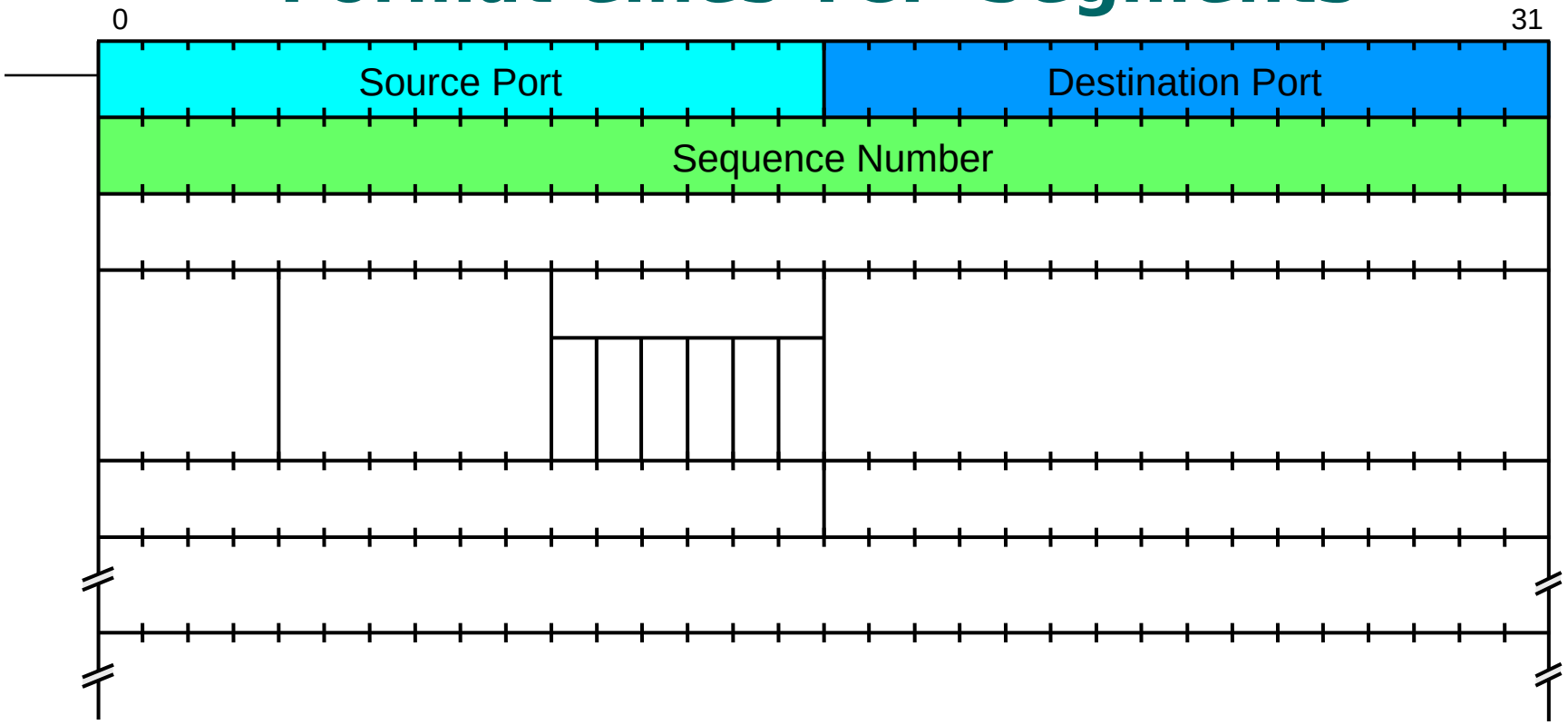
Source Port

Identifiziert die Anwendung auf der Senderseite.

Destination Port

Identifiziert die Anwendung auf der Empfängerseite.

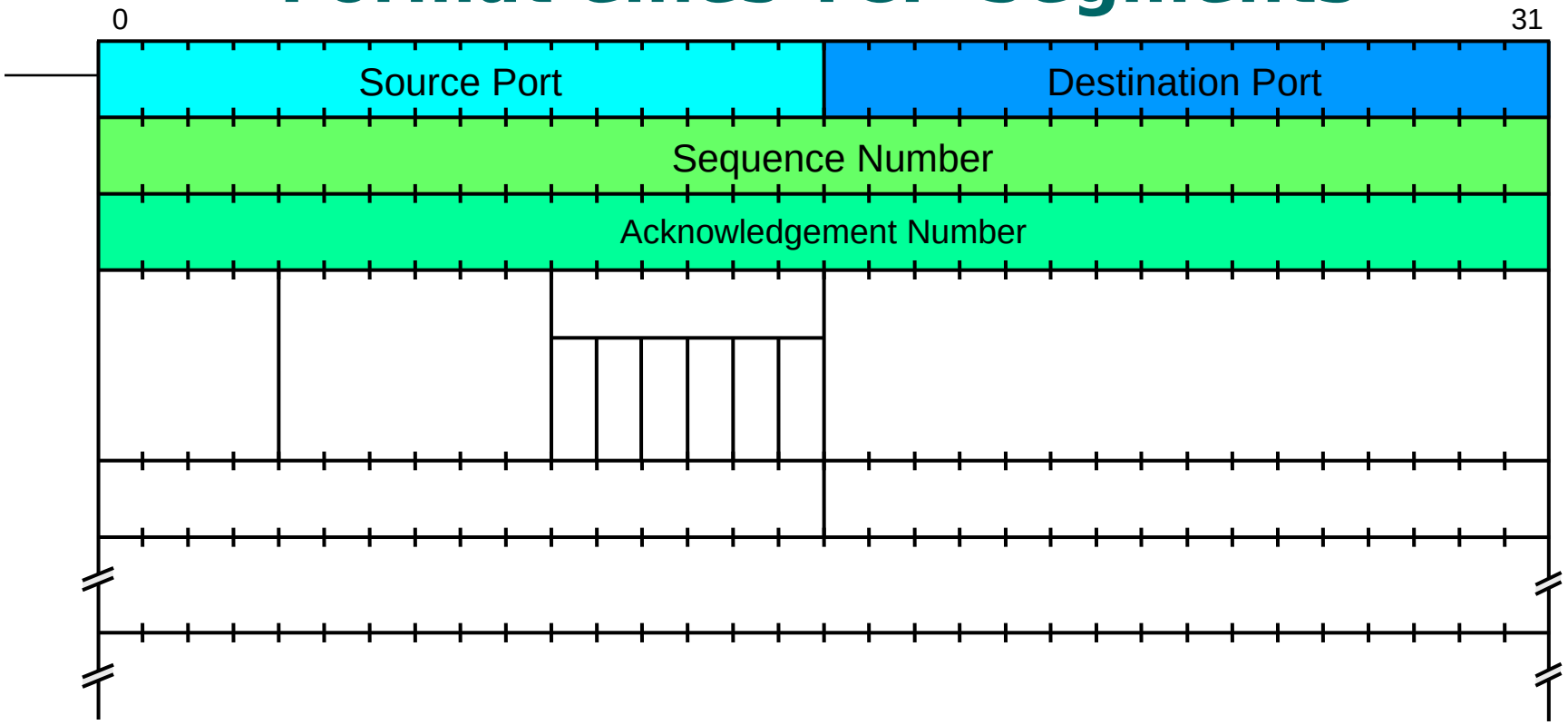
Format eines TCP-Segments



Sequence Number

TCP betrachtet die zu übertragenden Daten als nummerierten Byte-Strom.
Die *Sequence Number* ist die Nummer des ersten im Segment enthaltenen Datenbytes.

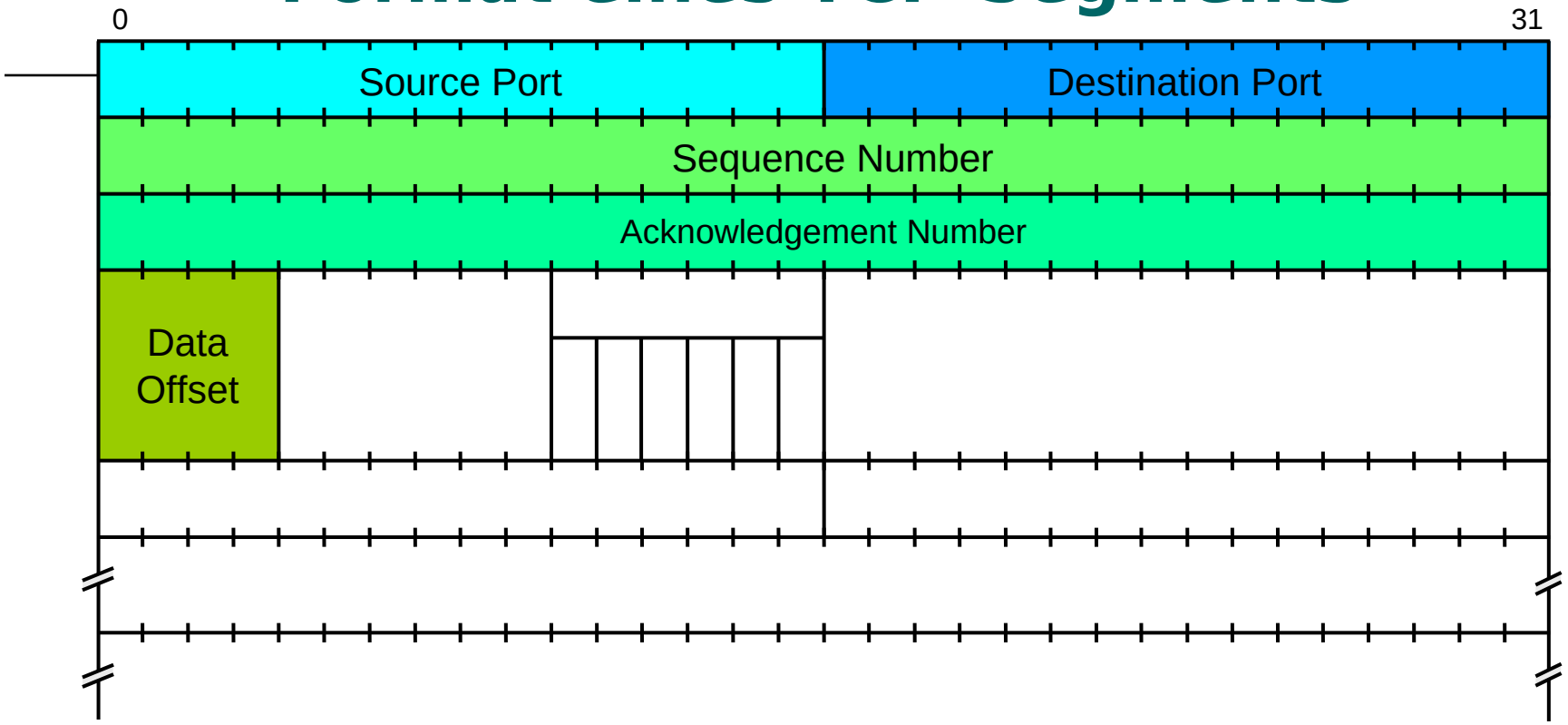
Format eines TCP-Segments



Acknowledgement Number

Dieses Feld bezieht sich auf einen Datenfluss in Gegenrichtung, d.h. hiermit werden Daten bestätigt, die die Station, die das Segment absendet, zuvor von der Zielstation empfangen hat.

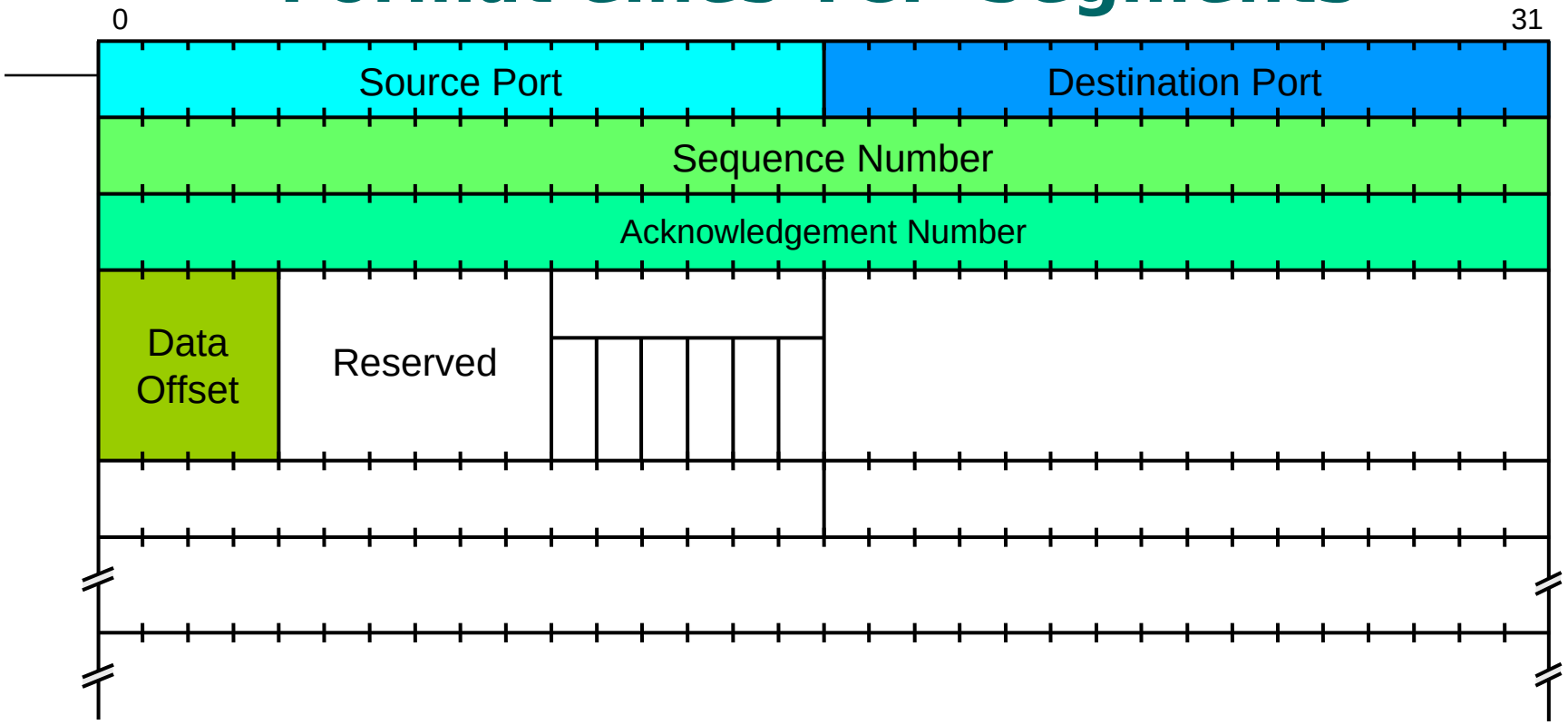
Format eines TCP-Segments



Data Offset

Da der Segment-*Header* Optionen enthalten kann, ist seine Länge nicht fix.
Im *Data Offset*-Feld wird die Länge (d.h. der Beginn des Datenteils) in 32-Bit-Einheiten angegeben.

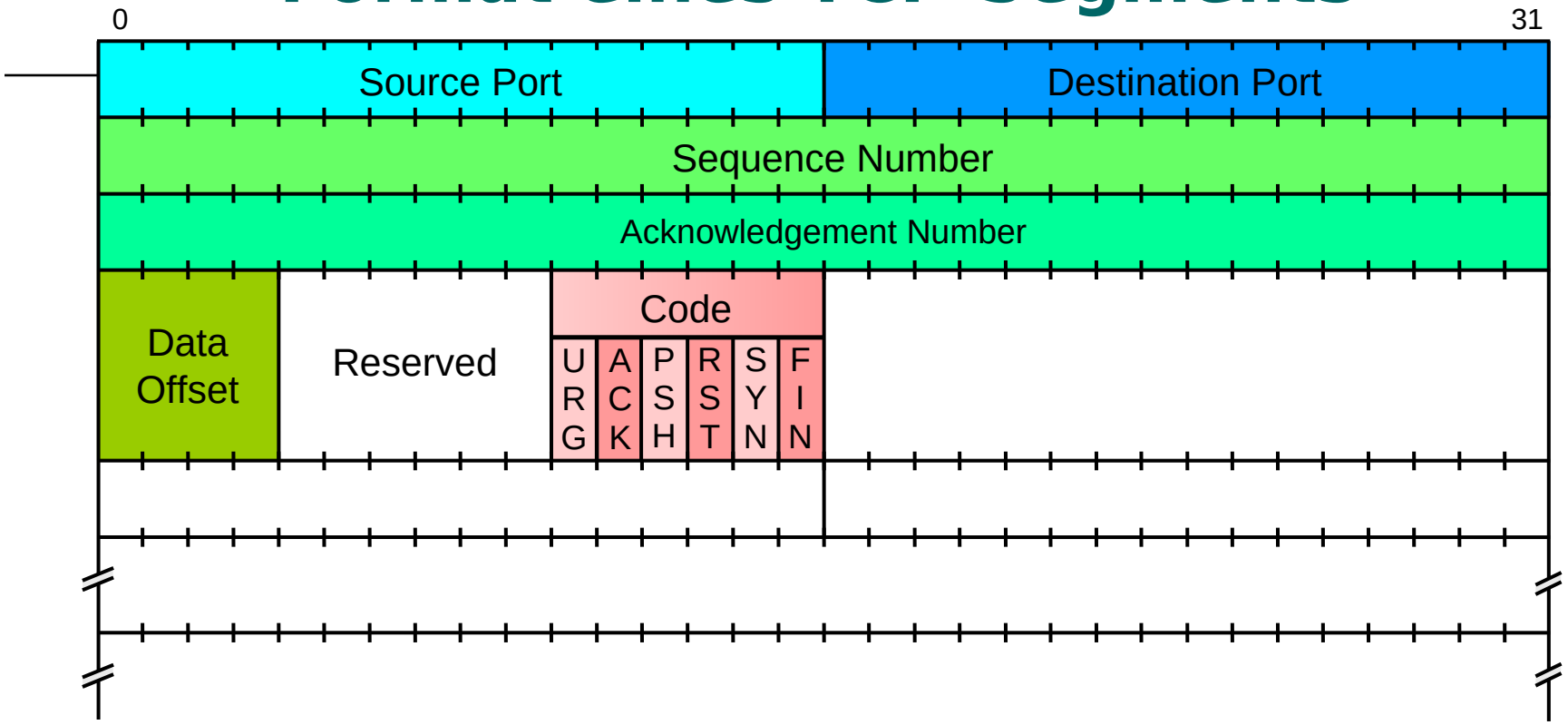
Format eines TCP-Segments



Res.

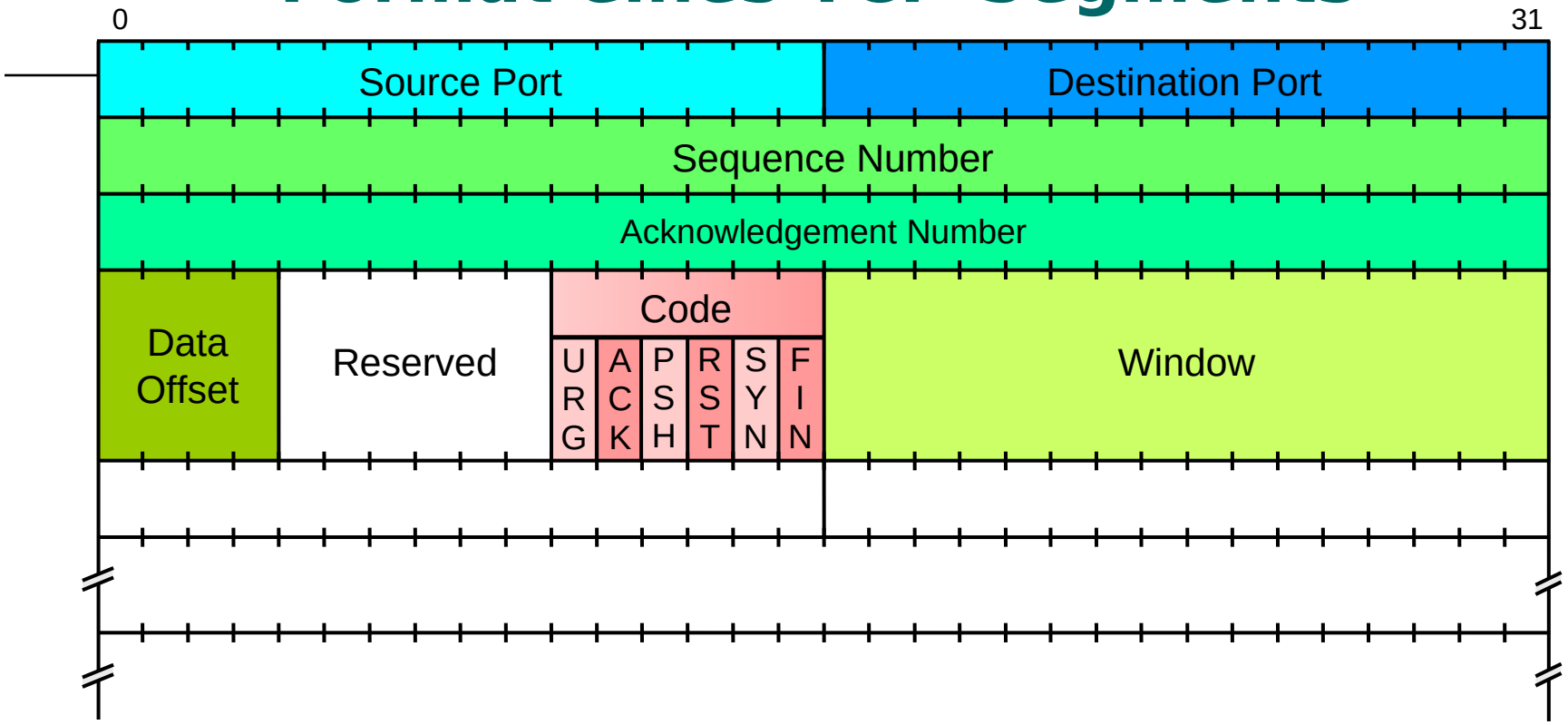
Reserviert für zukünftige Nutzung.

Format eines TCP-Segments



Code	Die Bits des Code-Feldes steuern besondere Funktionen des Segments.		
URG	*Urgent Pointer Field is valid*	ACK	*Acknowledgement Field is valid*
PSH	*This segment requests a push*	RST	*Reset the Connection*
SYN	*Synchronize sequence numbers*	FIN	*Sender has reached end of his byte stream*

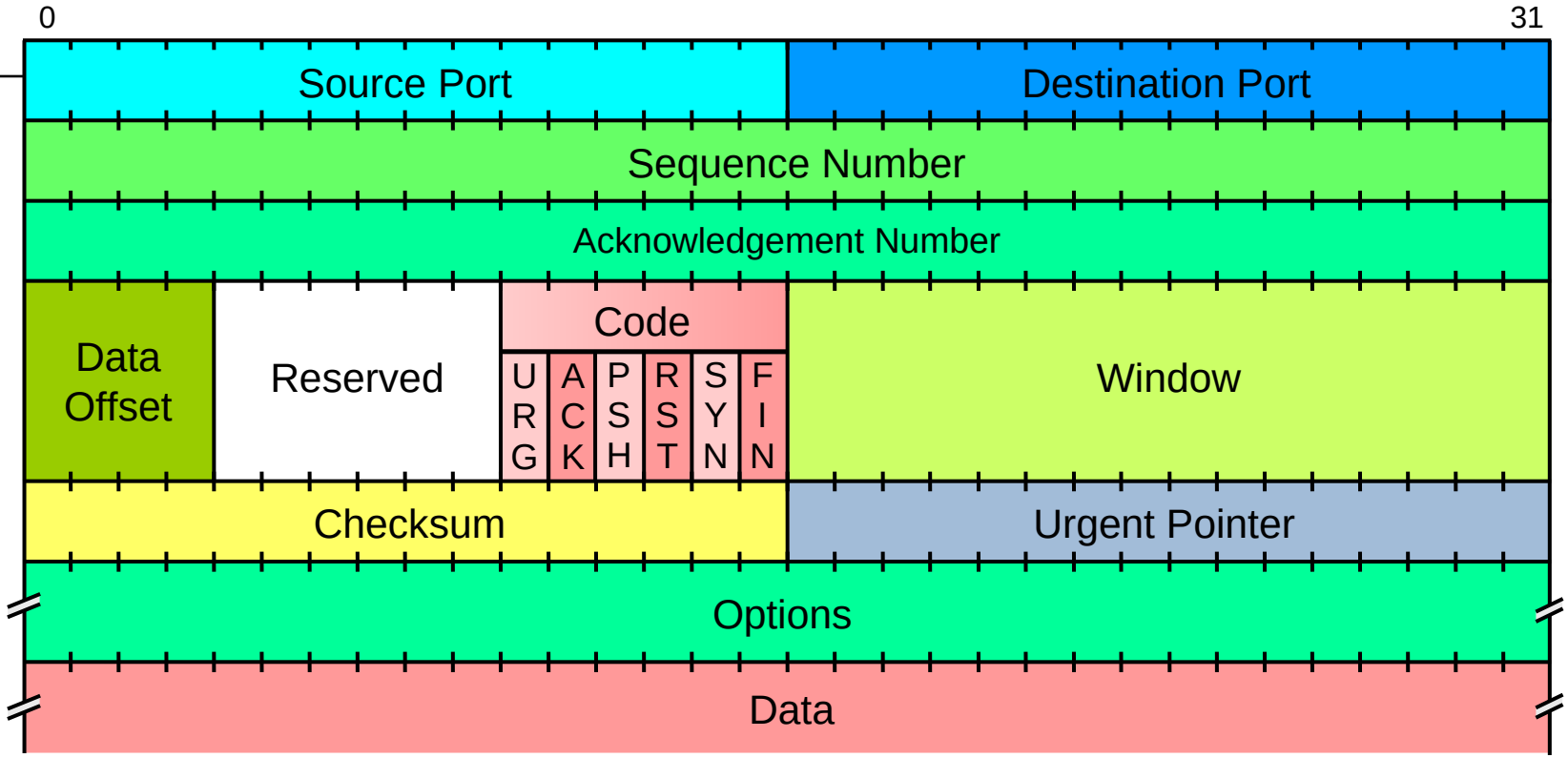
Format eines TCP-Segments



Window

Spezifiziert die Anzahl der Datenbytes (beginnend mit der im *Acknowledgement*-Feld angegebenen Byte-Nummer), die der Sender des Segments als Empfänger eines Datenstromes in Gegenrichtung akzeptieren wird.

Format eines TCP-Segments



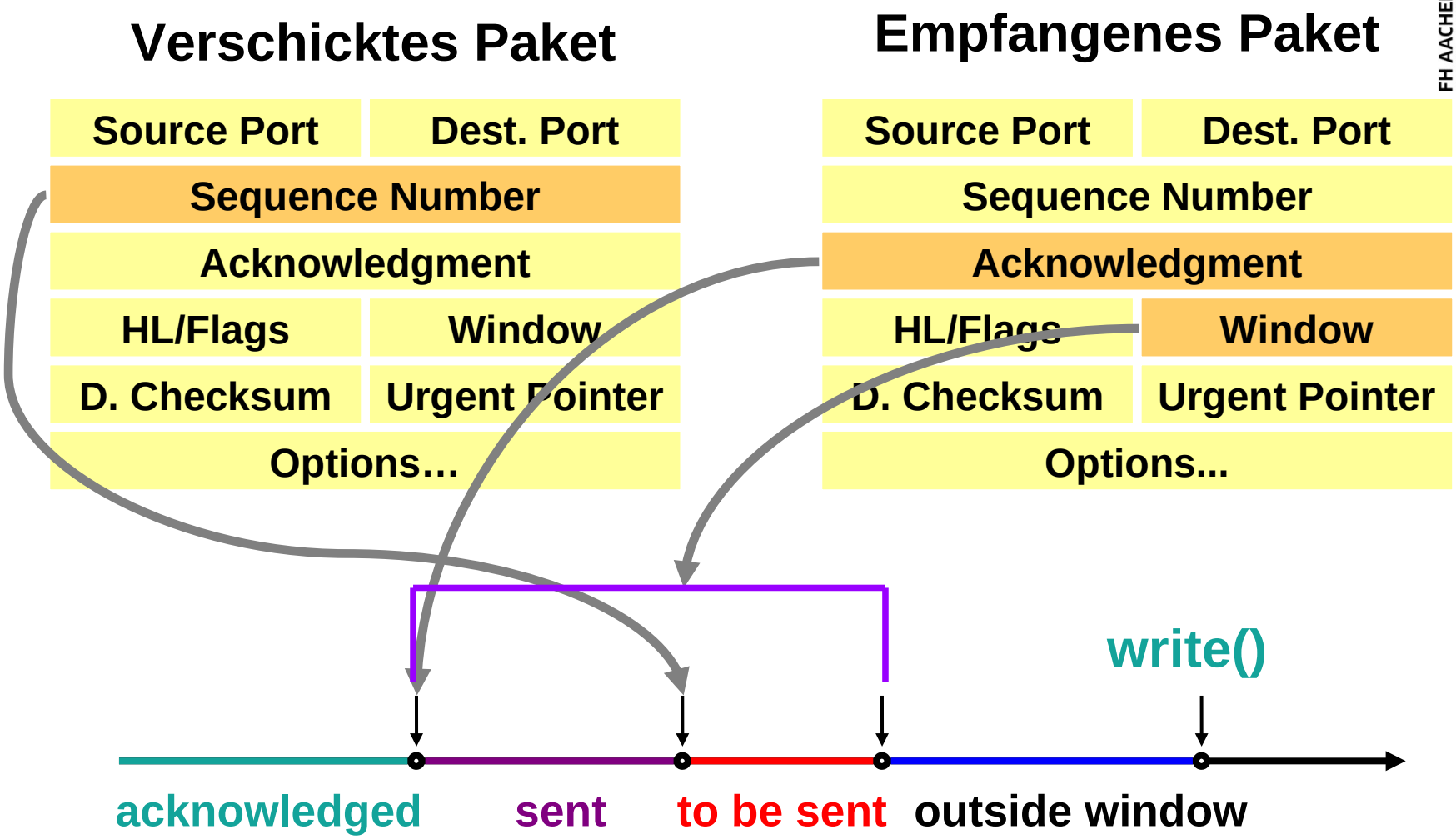
Checksum

16-Bit Längsparität über das gesamte Segment *(Header* + Daten).

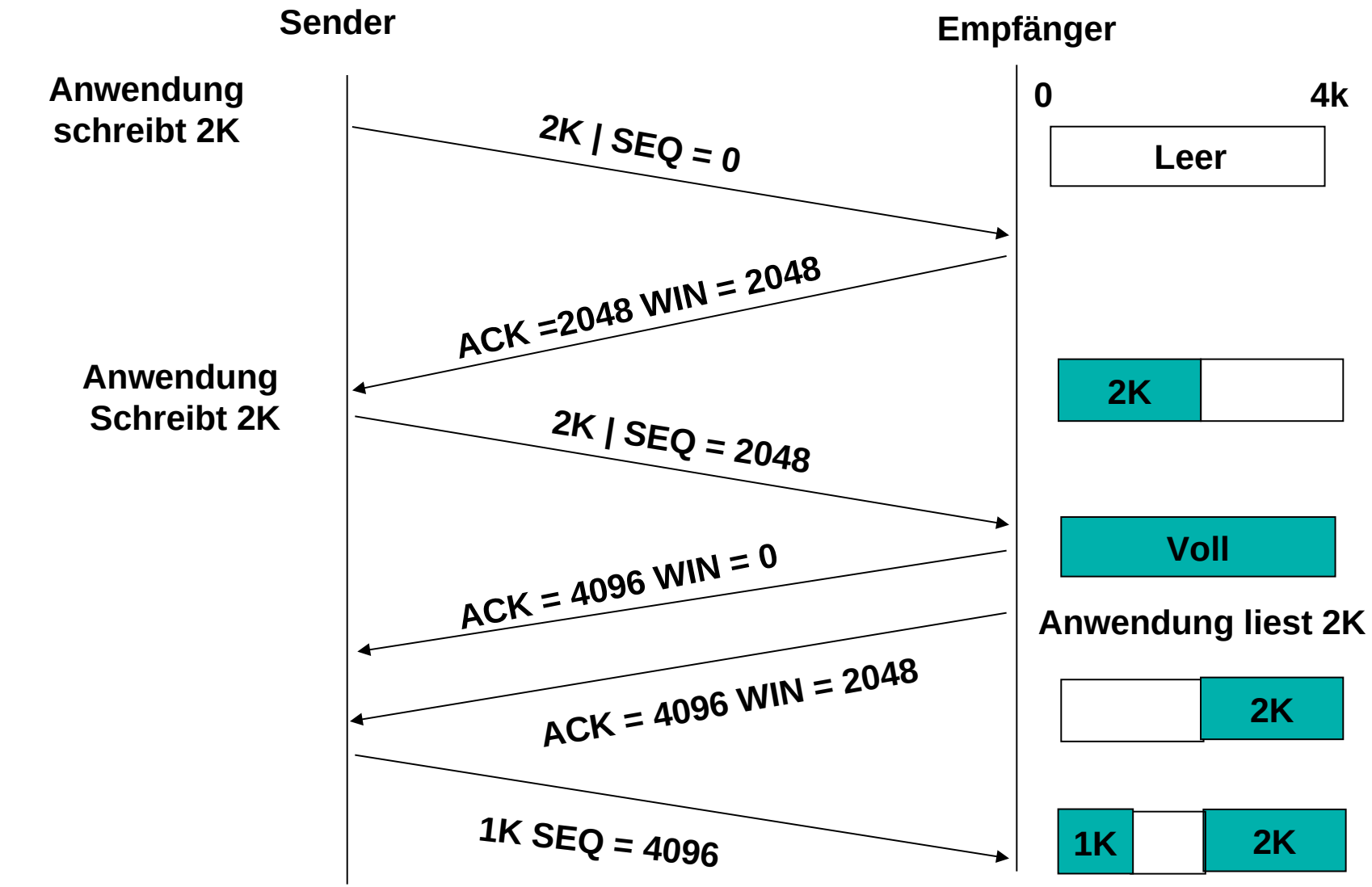
Urgent Pointer

Damit können Teile des zu übertragenden Byte-Stroms als dringend markiert werden.

Window-Flußkontrolle: Sender-Seite



Window-Management in TCP



Problemszenario 1:
Server antwortet nicht nach WIN=0

- Nach WIN=0 könnte das folgende ACK vom Empfänger der Daten verloren gehen…

- Auf Sender-Seite gibt es für diesen Fall einen weiteren Timer:

Transmission Control Protocol: Timer

- **Retransmission Timer**
  Timer wird beim Senden eines Segmentes gestartet.
  Bei Ablauf: Retransmission!

- **Persistence Timer**
  Timer wird nach Empfang einer Fenstergröße von 0 gestartet. Nach Ablauf Anfrage nach neuem Fenster (>0)

- **Keepalive Timer**
  Rel. lange Wartezeit. Wir bei jeder Transaktion rückgesetzt.
  Nach Ablauf → Abbau der Verbindung

- **Timer in TIME_WAIT**
  Folgt auf ein FIN innerhalb von 2 RTTs kein ACK, wird die Verbindung abgebaut!

Problemszenario 2: Bandwidth-Delay-Produkt >> 64kB

- Der TCP-Header stellt nur ein 16-bit-Feld für das Advertised Window zur Verfügung
- Damit kann zunächst keine Übertragungsfenster größer 64kB genutzt werden
- Das Bandwidth-Delay-Produkt gibt an, wie groß die Größe des Übertragungsfensters bei gegebener Bandbreite und Round-Trip-Zeit sein könnte:

  100ms * 1GBit = 12,5 MB >> 64 kB

- Wir brauchen größere ‚Fenster‘!
- **Lösung**: Window-Scale Option:

TCP Window Scale Option

- Einführen einer **Window-Scale-Erweiterung** die ein Skalierungsfaktor für das 16-Bit Window-Feld darstellt
- Der Skalierungsfaktor wird als neue TCP-Option eingeführt: Window Scale ist 3 Byte lang – Das letzte Byte gibt die Skalierung LOGARITHMISCH an (shift count: Das Fenster wird shift count bits nach links veschieben – hierbei ist der maximale Wert 14!)
- Die Option wird nur in einem SYN-Segement **beim Verbindungsaufbau übertragen**. Danach wird der Wert für die Ganze Verbindung angenommen
- Beide Seiten müssen eine Window-Scale-Option verschicken Null bedeutet "keine Skalierung"
- Die neue maximale Fenstergröße in Bytes errechnet sich wie folgt:

$$65536 * 2^{14} = 65535 * 16384 = 1.073.725.440$$

- TCP-intern wird die Fenstergröße jetzt als 32-Bit-Zahl verwaltet

TCP Window Scale Option: RFC 1323

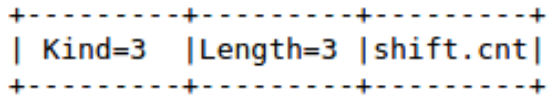
but can be overridden by a user program before a TCP connection is opened. This determines the scale factor, and therefore no new user interface is needed for window scaling.

2.2 Window Scale Option

The three-byte Window Scale option may be sent in a SYN segment by a TCP. It has two purposes: (1) indicate that the TCP is prepared to do both send and receive window scaling, and (2) communicate a scale factor to be applied to its receive window. Thus, a TCP that is prepared to scale windows should send the option, even if its own scale factor is 1. The scale factor is limited to a power of two and encoded logarithmically, so it may be implemented by binary shift operations.

TCP Window Scale Option (WSopt):

Kind: 3 Length: 3 bytes



TCP Window Scale Option

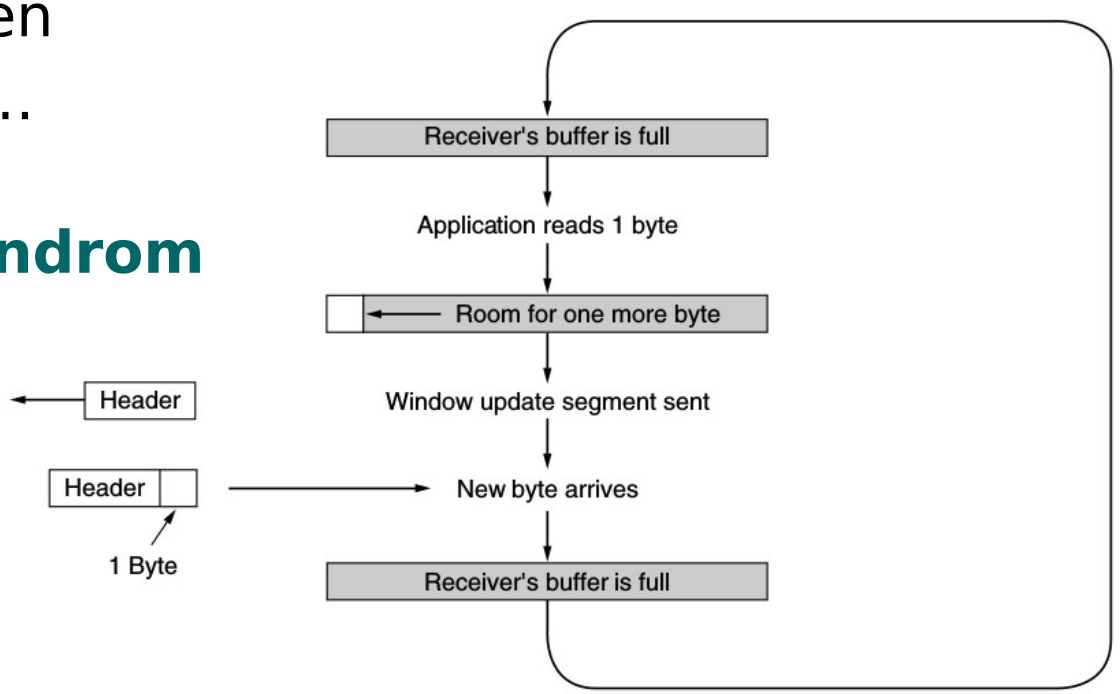
- Ohne Window-Scale-Option ist TCP nicht geeignet für sogenannte Long-Fat-Pipes
- Die Window-Scale Option wird beim Aufbau der Verbindung **in beiden Richtungen** separat mitgeteilt
- Anschaulich handelt es sich um die Anzahl der binären Nullen, die hinter dem Advertised/Receiver Window zusätzlich stehen
- Damit können aber auch nur noch größere Änderungen im Puffer des Empfängers angezeigt werden
- Verpasst eine Anwendung das Setzen der Option vor einer Verbindung, so kann sie ggf. die gewünschte Rate nicht erzielen

Problemszenario 3: Pakte und einzelne Zeichen

- Das Versenden und Empfangen von Daten geschieht in Paketen von z.B. typischerweise 1460 Bytes:
- Der **Sender** wartet erst, bis er die Daten für solch ein Paket gesammelt hat, und sendet dann das ganze Paket
- Der **Empfänger** würde erst die Daten zwischenspeichern, bevor er sie der Applikation zur Verfügung stellt.

Problemszenario 3: Pakte und einzelne Zeichen

- Was passiert, wenn das Empfangsfenster ‚voll' ist, und die empfangende Applikation die Zeichen aber einzeln ausliesst?
- → Der Sender würde für jedes einzelne Zeichen ein neues Paket senden
- → Sehr ineffizient…
- → Name dafür:
  Silly Window Syndrom
- Lösung (Clark): Empfänger darf neue Fenstergröße erst senden, wenn in seinem Buffer mehr Platz ist (z.B. ein Segment)

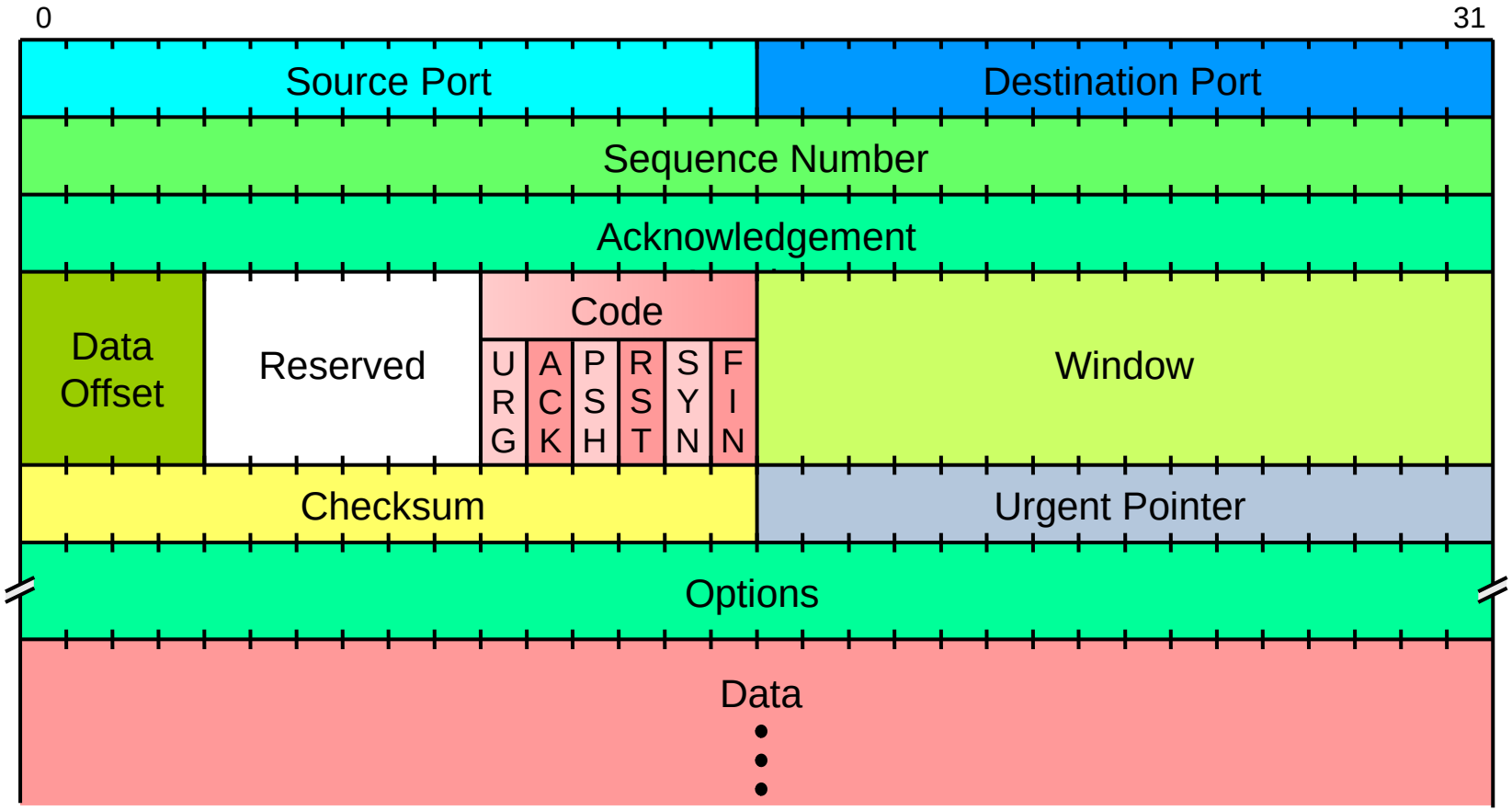


Problemszenario 3: Pakte und einzelne Zeichen

- Was passiert, wenn der Sender wichtige Daten hat, die er ohne Pufferung an die empfangende Applikation weiter geben will?
- Beispiel: Ctrl-C zum Abbruch einer Ausgabe

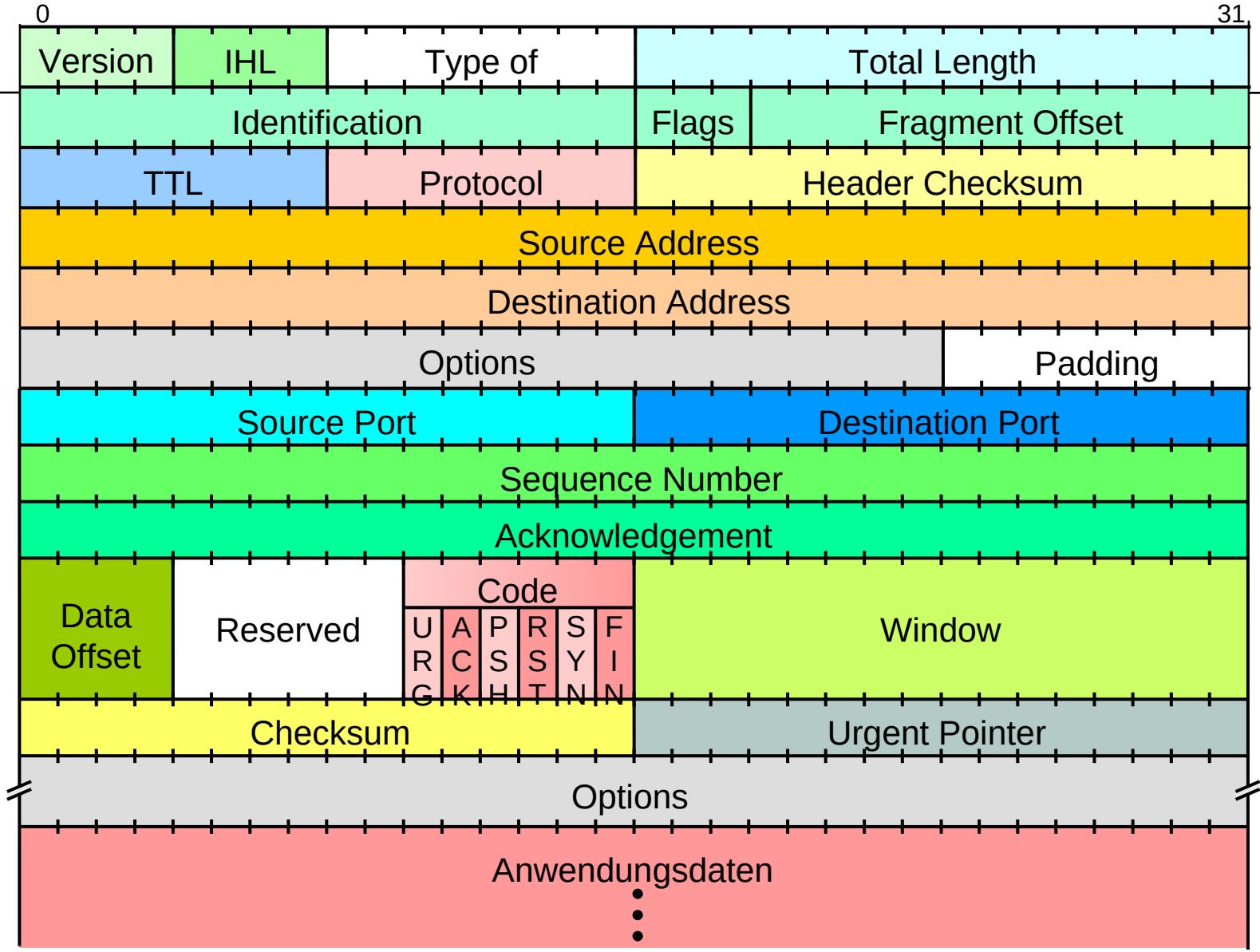
- Lösung:
  - **PSH** – Flagge im TCP Header:
    Daten auf Empfängerseite direkt zustellen (kein Puffern)
- Alternative:
  - **URG** – Flagge im TCP Header:
    ‚Event' auf Empfängerseite auslösen

Format eines TCP-Segments



Frage: Wie wird die Länge eines Datagramms ermittelt?

Was kommt am Rechner an?



FH Aachen
Fachbereich 9 Medizintechnik und Technomathematik
Prof. Dr.-Ing. Andreas Terstegge
Straße Nr.
PLZ Ort
T +49. 241. 6009 53813
F +49. 241. 6009 53119
Terstegge@fh-aachen.de
www.fh-aachen.de