DHBW Karlsruhe
Duale Hochschule Baden-Württemberg

Netztechnik I

T2INF4201.1

Transportschicht
(Transport Layer)

Markus Götzl
Dipl.-Inform. (FH)
mail@markusgoetzl.de

http://www.dhbw-karlsruhe.de

Transportschicht (Transport Layer)

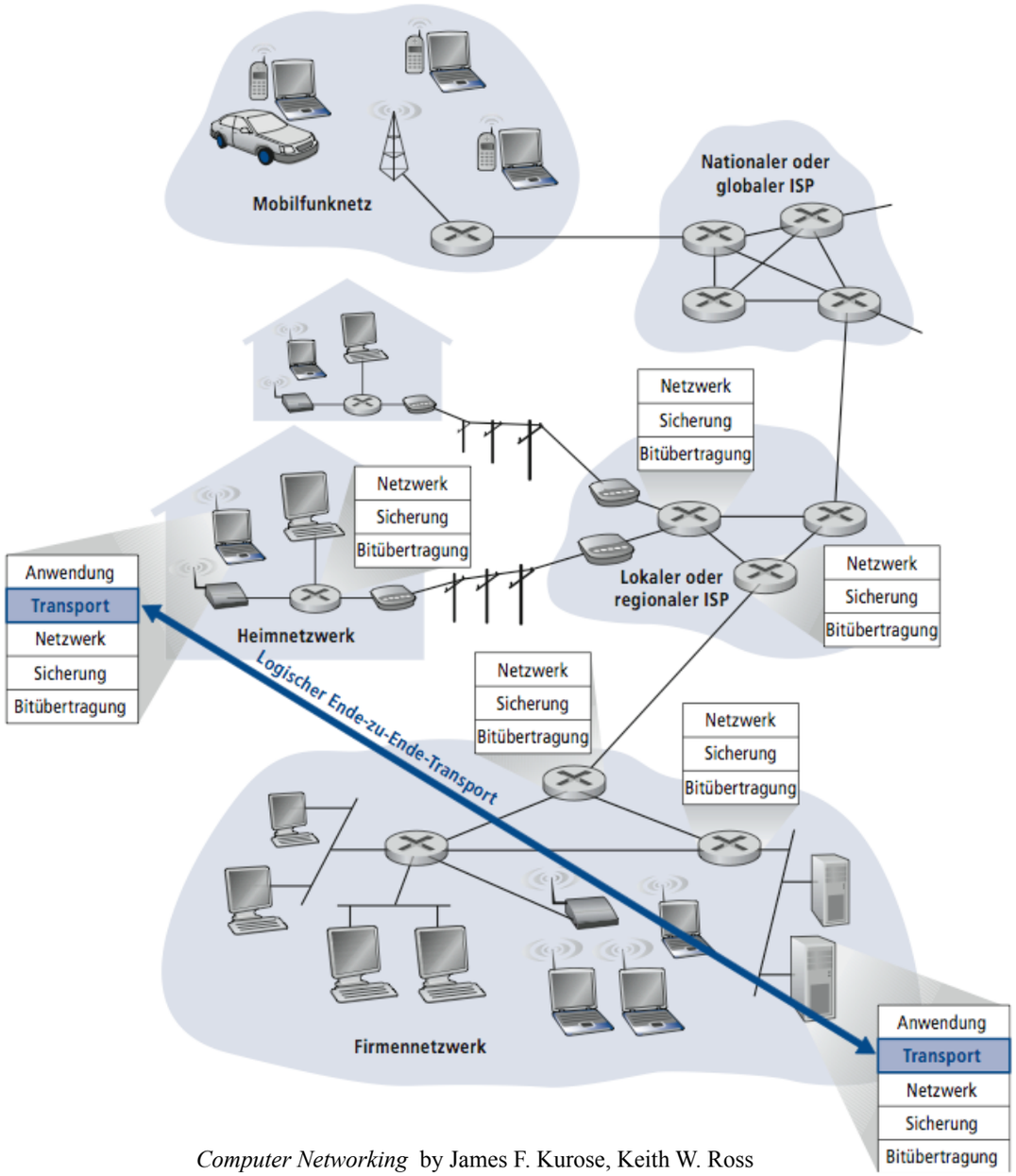
Verbindungsaufbau interessant

- Aufgaben der Transportschicht
- Dienste der Transportschicht
- Transportschicht Dienstelemente
- Transportschicht im Internet (UDP/TCP)

Aufgaben der Transportschicht

- Transport von Paketen von der Quelle zum Ziel (Ende-zu-Ende).
  - In Abgrenzung zur Vermittlungsschicht wird die dafür erforderliche Logik auf den Rechnern der Benutzer ausgeführt und nicht wie in der Vermittlungsschicht auf Maschinen der Netzwerkprovider (Routern).
- Zuverlässiger Transport von Paketen unabhängig von der Zuverlässigkeit des aktuellen physikalischen Netzwerks.
- Bereitstellung einer Abstraktionsebene für Applikationen zur Nutzung eines Netzwerks.

Aufgaben der Transportschicht

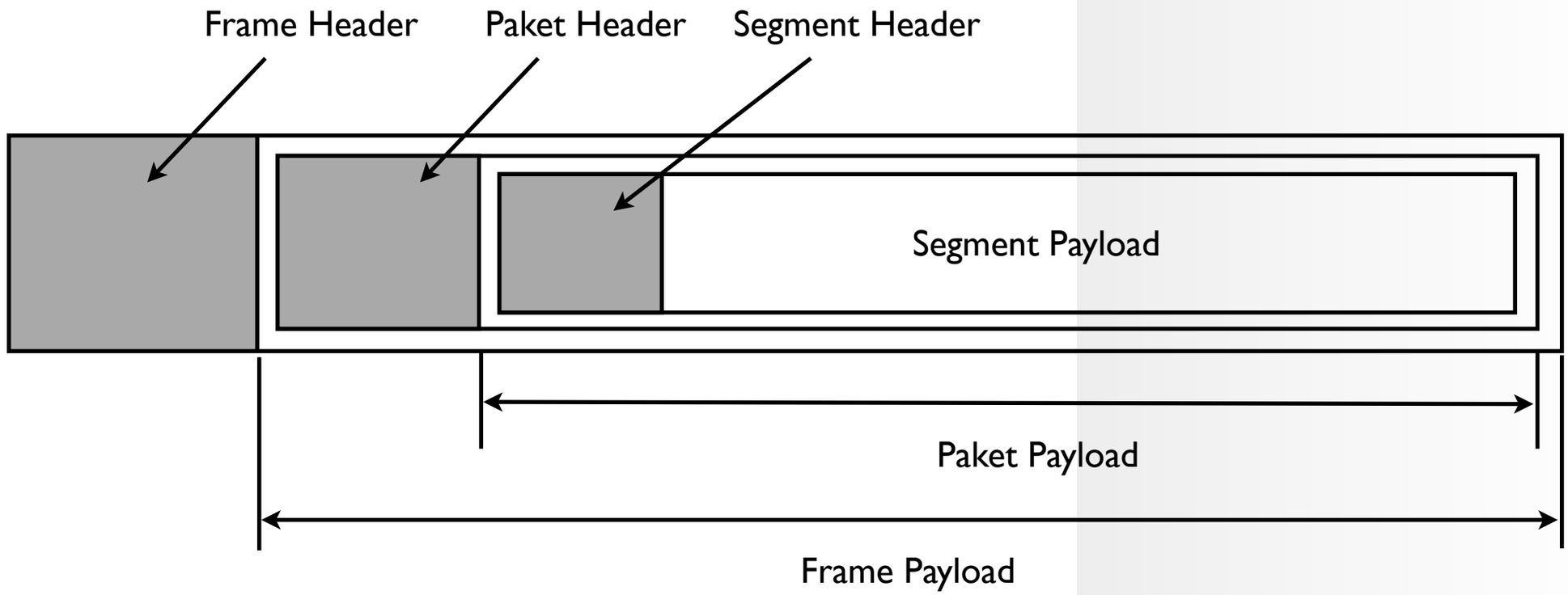


Computer Networking by James F. Kurose, Keith W. Ross

- Die Transportschicht realisiert einen logischen Ende-zu-Ende-Transport zwischen Anwendungsprozessen

Aufgaben der Transportschicht

Segmente, Pakete und Frames



Dienste der Transportschicht

- Verbindungsorientierter Dienst
  - Es wird ein zuverlässiger verbindungsorientierter Transport von Daten auf “echten” Netzwerken d.h. Fehlerbehafteten Netzwerken angeboten. Dies wir durch eventuelle Neuanforderungen von Fehlerbehafteten Daten erreicht.
  - Beispiel: TCP
- Verbindungsloser Dienst
  - Es wird der bestmögliche Transport von Daten, abhängig vom Netzwerk, angeboten (Best Effort). Das bedeutet, das weder eine feste Verbindung zwischen Sender und Empfänger aufgebaut wird noch, dass Anstrengungen unternommen werden um Datenverluste zu erkennen bzw. zu kompensieren.
  - Dies kann sinnvoll sein wenn Zeitverzögerung eine höheren Stellenwert einnimmt als Zuverlässigkeit.
  - Beispiel: UDP

Transportschicht Dienstelemente

- Basis Dienstelemente welche von der Transportschicht zur Verfügung gestellt werden (Verbindungsorientiert)
  - Dienstelemente der Transportschicht befinden sich auf den Rechnern der Benutzer.

Dienstelement	Versendete Pakete	Beschreibung
LISTEN	(keine)	Warten auf eine Verbindungsanfrage
ACCEPT	CONNECTION ACC	Verbindung akzeptieren
CONNECT	CONNECTION REQ	Senden einer Verbindungsanfrage
SEND	DATA	Senden der Informationen
RECEIVE	(keine)	Warten bis eine DATA Pakete empfangen wird
DISCONNECT	DISCONNECTION REQ	Anfrage zum Beenden der Verbindung

Transportschicht Dienstelemente

Dienstelement	Versendete Pakete	Beschreibung
LISTEN	(keine)	Warten auf eine Verbindungsanfrage
ACCEPT	CONNECTION ACC	Verbindung akzeptieren
CONNECT	CONNECTION REQ	Senden einer Verbindungsanfrage
SEND	DATA	Senden der Informationen
RECEIVE	(keine)	Warten bis eine DATA Pakete empfangen wird
DISCONNECT	DISCONNECTION REQ	Anfrage zum Beenden der Verbindung

- Beispielhafte Client-Server Kommunikation

1. Client sendet CONNECT
2. Server sendet ACCEPT
3. Datenaustausch über SEND bzw. RECEIVE
4. Client bzw. Server sendet DISCONNECT
    - DISCONNECT kann symmetrisch oder asymmetrisch sein.
        - symmetrisch: Beide Kommunikationspartner müssen ein DISCONNECT senden
        - asymmetrisch: Ein Kommunikationspartner sendet ein DISCONNECT

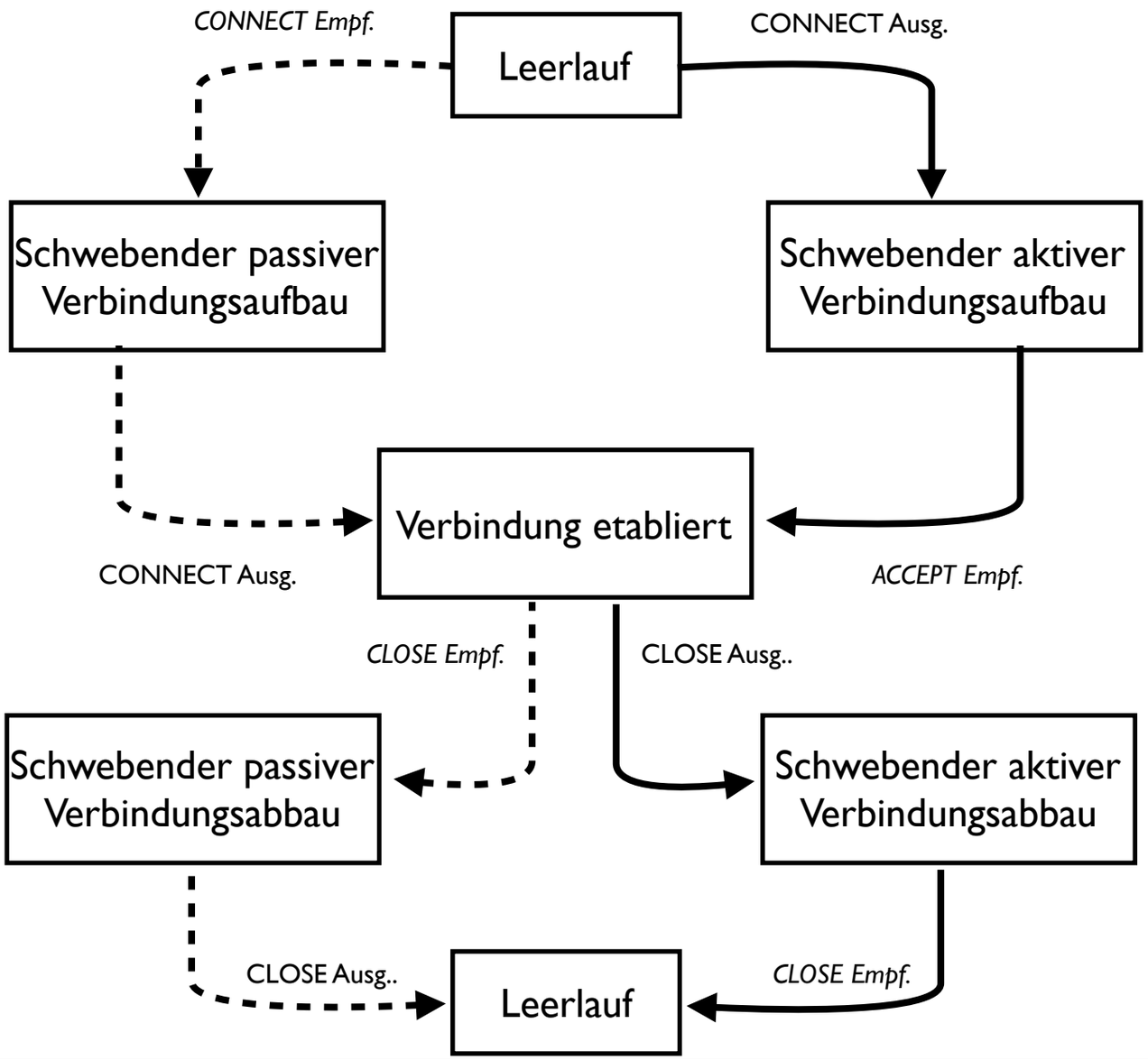
Berkely Sockets

- Sockets dienen zur Adressierung (Portnummer) und als Schnittstelle der Transportschicht im Internet (TCP/UDP)
  - Die Portnummer ist sowohl im TCP- als auch im UDP-Header definiert und 16 Bit groß. Damit sind 2^16 = 65536 Ports möglich.
  - **Well Known Ports:** *0-1023* sind bestimmten Protokollen zugeordnet und werden von der IANA (Internet Assigned Numbers Authority) verwaltet.
  - **Registered Ports:** *1024 - 49.151* sind ebenfalls von der IANA verwaltet, können aber selbst verwendet werden.
  - **dynamische bzw. private Ports:** *49152 – 65535* diese werden hauptsächlich als Client-Ports verwendet und dynamisch bei Bedarf allokiert.

Berkely Sockets - Schnittstelle

Dienstelement	Beschreibung
SOCKET	Einen neuen Kommunikationsendpunkt erstellen
BIND	Eine neue lokale Adresse mit einem Socket assoziieren
LISTEN	Verbindungsanfragen akzeptieren
ACCEPT	Eine passive eingehende Verbindung erstellen
CONNECT	Aktiv eine Verbindung erstellen
SEND	Daten über eine Verbindung senden
RESEIVE	Daten von einer Verbindung empfangen
CLOSE	Die Verbindung abbauen

Berkely Sockets - Zustandsdiagramm



SOCKET: Erzeugt einen neuen Kommunikationsendpunkt und allokiert den Benötigten Speicher (Erzeugt einen file descriptor)

BIND: Ordnet einem neu erzeugten Socket eine Netzwerkadresse zu (IP + Portnummer)

LISTEN: Allokiert Speicher für eingehende Verbindungen (nicht Blockierend - mehrere Verbindungen möglich)

ACCEPT: Erzeugt einen neuen Socket (Server) + file descriptor wird erzeugt. Der Server kann z.B. einen neuen Prozess/Thread erzeugen und am alten Socket weiter auf eingehende Verbindungen warten.

SEND/RESEIVE: Austausch der Daten

CLOSE: Socket und Speicher werden freigegeben.Wenn Server und Client CLOSE ausgeführt haben ist die Verbindung beendet (symmetrisch).

Empfang eines Dienstelementes

Ausführen eines Dienstelementes

Client Statusübergang

Server Statusübergang

Berkely Sockets - Ablauf

- Stream Sockets (TCP)
  - **Client**
    1. Socket erstellen
    2. erstellten Socket mit der Server-Adresse verbinden, von welcher Daten angefordert werden sollen
    3. senden und empfangen von Daten
    4. evtl. Socket herunterfahren (shutdown())
    5. Verbindung trennen, Socket schließen
  - **Server**
    6. Server-Socket erstellen
    7. binden des Sockets an eine Adresse (Port), über welche Anfragen akzeptiert werden
    8. auf Anfragen warten
    9. Anfrage akzeptieren und damit ein neues Socket-Paar für diesen Client erstellen
    10. bearbeiten der Client-Anfrage auf dem neuen Client-Socket
    11. Client-Socket wieder schließen.

Berkely Sockets - Ablauf

- Datagram Sockets (UDP)
  - **Client**
    1. Socket erstellen
    2. An Adresse senden
  - **Server**
    3. Socket erstellen
    4. Socket binden
    5. warten auf Pakete

Transmission Control Protocol (TCP)

- TCP bietet eine Zuverlässige Verbindung zwischen zwei Kommunikationspartnern.
- Ursprüngliche Definition in RFC 793. Anpassungen und Verbesserung u.a. in den RFCs 1122, 1323, 2018, 2581, ...
- RFC 4614 fasst die Änderungen zusammen und verweist auf die verschiedenen RFCs.
- TCP verwendet Sockets, die definieren, welche Anwendung für welchen Pakete zuständig ist (Port Adresse).

Transmission Control Protocol (TCP)

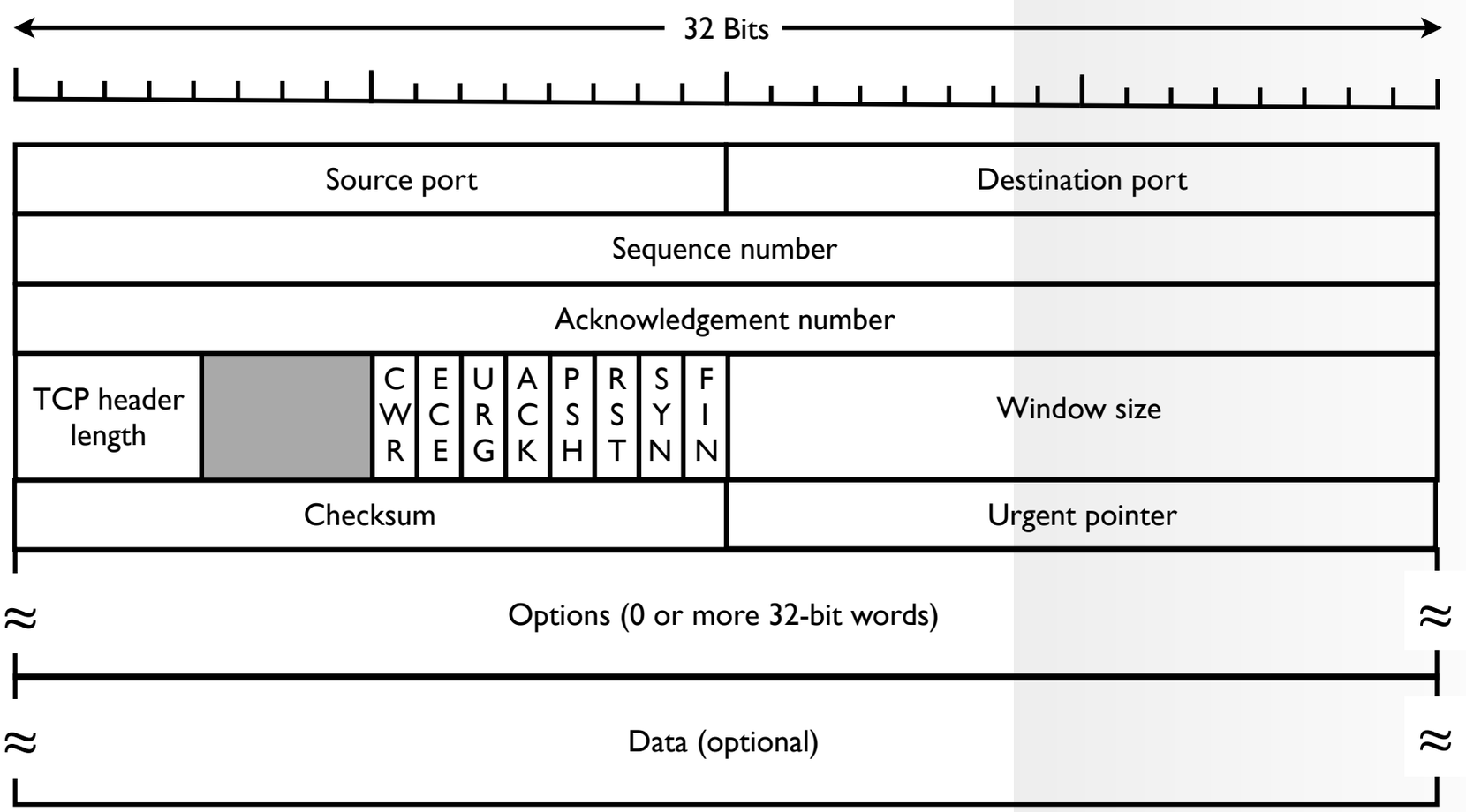
- TCP definiert zwei Typen von Segmenten:
  - *Control-Segmente*: Diese Segmente enthalten nur Steuerinformationen, z.B. Handshakes oder Keep Alive (Kein Payload/Daten).
  - *Payload-Segmente*: Zusätzlich zu den Steuerinformationen im Segment sind hier auch Nutzdaten höherer Schichten enthalten.

Transmission Control Protocol (TCP)

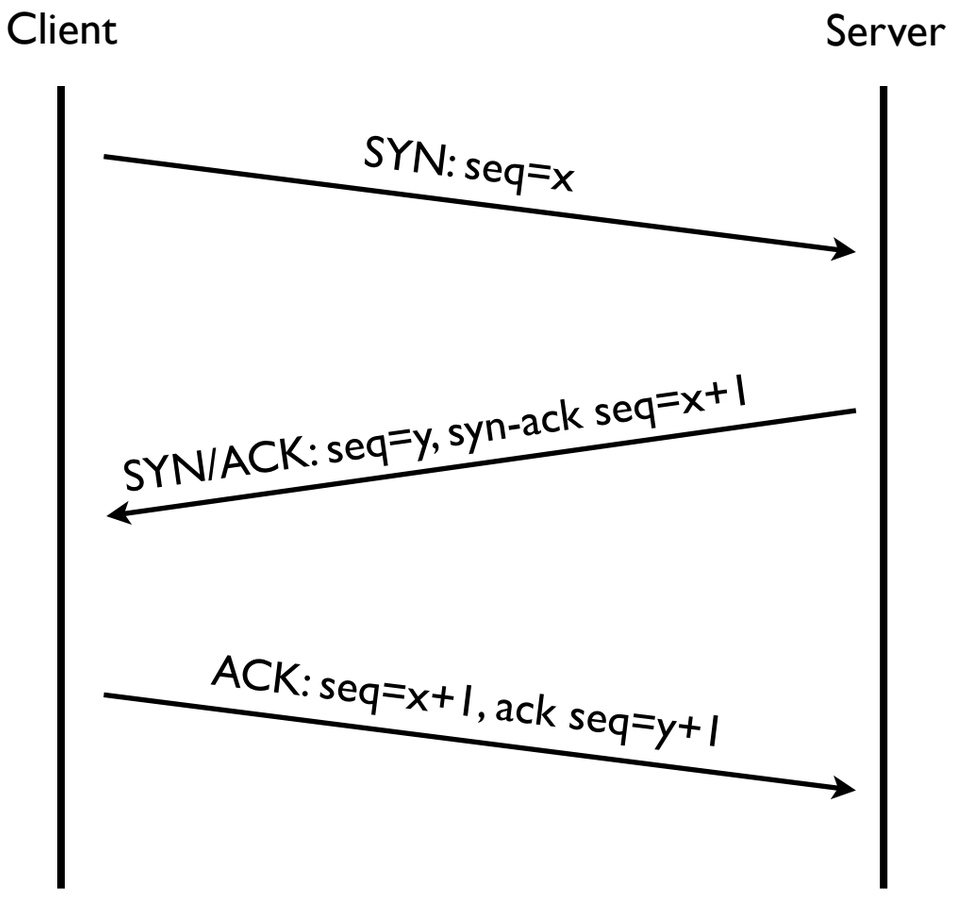
- Eigenschaften von TCP:
  - Datenintegrität und Zuverlässigkeit
    - Die Integrität der Daten wird mittels der Prüfsumme im Paketkopf geprüft während die Reihenfolge durch Sequenznummern sicher gestellt wird. Der Sender wiederholt das Senden von Paketen, falls keine Bestätigung innerhalb einer bestimmten Zeitspanne (Timeout) eintrifft.
  - Duplexbetrieb
    - Sender und Empfänger können abwechselnd über die gleiche Verbindung senden
  - Überlastkontrolle (Congestion Control)
    - Die Verlustrate wird von einem IP-Netzwerk ständig beobachtet. Abhängig von der Verlustrate wird die Senderate durch geeignete Algorithmen beeinflusst: Normalerweise wird eine TCP/IP-Verbindung langsam gestartet (Slow-Start) und die Senderate schrittweise erhöht, bis es zum Datenverlust kommt. Ein Datenverlust verringert die Senderate, ohne Verlust wird sie wiederum erhöht. Insgesamt nähert sich die Datenrate so zunächst dem jeweiligen zur Verfügung stehenden Maximum und bleibt dann ungefähr dort. Eine Überbelastung wird vermieden.

Transportschicht im Internet (UDP/TCP)

TCP Header Übersicht:



TCP - Verbindungsaufbau



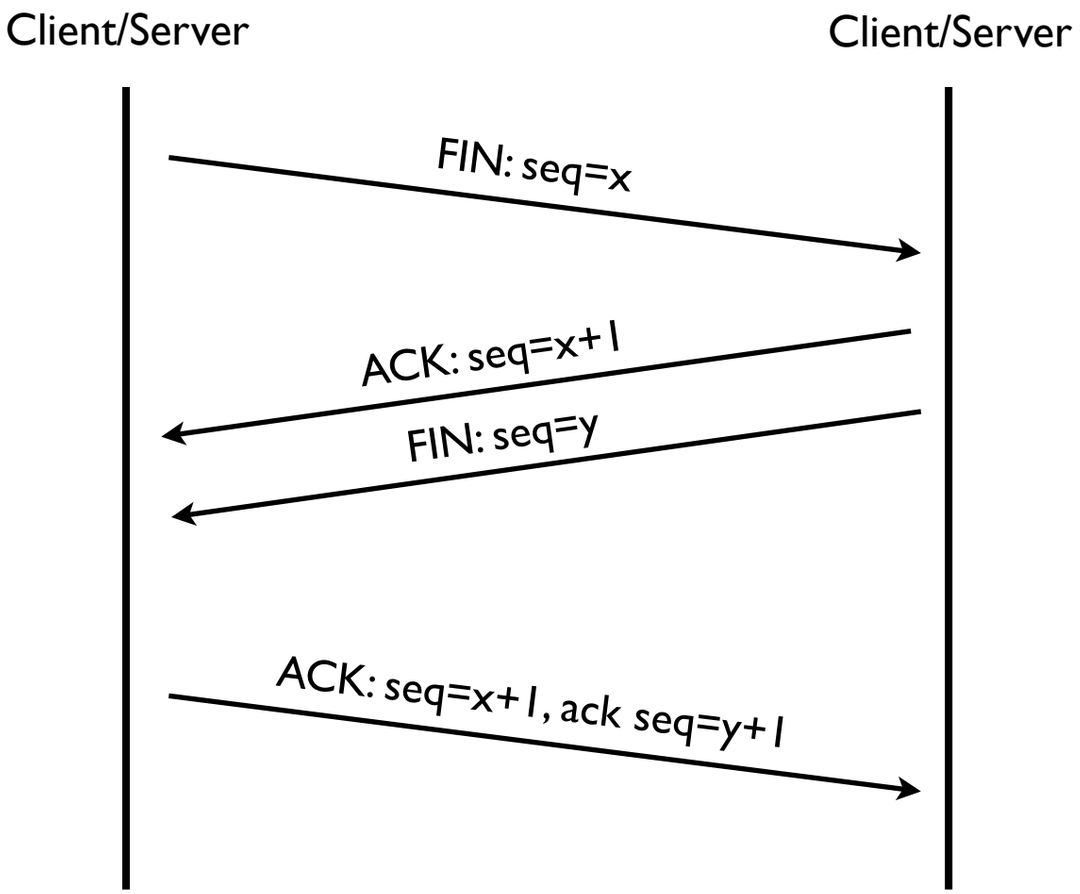
- 3-Wege-Handshake

1. Der Client sendet ein Control-Segment, dessen SYN-Flag gesetzt ist (SYN-Segment). Zusätzlich erzeugt er eine **Syn-Sequenznummer(x)**. Die Start-Sequenznummer ist eine beliebige Zahl, deren Generierung von der jeweiligen TCP-Implementierung abhängig ist. Sie sollte jedoch möglichst zufällig sein.

2. Der Server bestätigt er den Erhalt des ersten SYN-Segments und stimmt dem Verbindungsaufbau zu, indem er ein SYN/ACK-Segment zurückschickt (SYN und ACK-Flag gesetzt). Dieses Segment enthält eine **Syn-Ack-Sequenznummer** welche der **Syn-Sequenznummer(x)+1** entspricht und eine **neue Sequenznummer(y)** welche ebenfalls zufällig erzeugt wird.

3. Der Client bestätigt den Erhalt des SYN/ACK-Segments durch das Senden eines eigenen ACK-Segments dieses enthält die **Sequenznummer(x)+1** und eine **Ack-Sequenznummer(y)+1**

TCP - Verbindungsabbau



- Verbindungsabbau
  - Beim Abbau einer Verbindung muss sichergestellt werden, dass beide Parteien sich auf eine Beendigung der Verbindung geeinigt haben (symmetrisch).
  - Der Abbau erfolgt in einem 4-Wege-Handshake, bei dem beide Parteien sowohl ein FIN senden als auch ein ACK erwarten (Bei Kombination der ACK und FIN Segmente ist auch ein 3-Wege-Handshake möglich).
  - Sollten auf die FIN Segmente keine ACKs erfolgen, wird die Übertragung wiederholt und nach einem Timeout aufgegeben und die Verbindung einseitig beendet.

TCP - Verbindungsabbau - Two-Army-Problem

- Da eine gegenseitige Bestätigung der FIN-Segmente gefordert ist entsteht ein unlösbares Problem, dass “Two-Army-Problem”
  - Anschaulich lässt sich dieses Problem mit der Annahme beschreiben, dass eine sehr starke Armee (weiß) in einem Tal von zwei schwächeren Armeen (blau) eingeschlossen ist.
  - Sollte eine einzelne blaue Armee die weiße angreifen, wird sie verlieren, sollten aber beide blaue Armeen gleichzeitig angreifen, dann ist ein Sieg gewiss.
  - Beide blaue Armeen müssen daher einen Zeitpunkt absprechen
  - Leider ist es den Armeen nur möglich einen Kurier durch das Tal zu schicken, wodurch das Risiko besteht, dass der Kurier abgefangen wird und die Botschaft nicht ankommt.
    - Die **Armee Blau 1** sendet also einen Kurier und fragt: “Angriff, morgen 12 Uhr, in Ordnung?”
    - Nehmen wir an, der Kurier gelangt **erfolgreich durch das Tal zu Armee Blau 2** und bringt ebenfalls **erfolgreich die Bestätigung zurück**.
    - dummerweise kann sich **die Armee Blau 2 nicht sicher sein, dass die Bestätigung tatsächlich ankam**, und sie evt. alleine angreift.
    - Dies weiß auch die **Armee Blau 1 und wird selbst mit Bestätigung nicht angreifen**.

User Datagram Protocol (UDP)

- UDP ist ein sehr einfaches Protokoll. Es verzichtet auf Zuverlässigkeit zu Gunsten von Einfachheit und Geschwindigkeit. Es kann keine Garantie geben, dass ein gesendetes Paket auch tatsächlich zugestellt wurde. Dies überlässt UDP (wie auch schon IP) höheren Schichten.
- UDP wurde 1980 im RFC 768 spezifiziert.
- UDP wird eingesetzt wenn Antwortzeiten eine Rolle spielen (DNS) und/oder einzelne Pakete gefahrlos verloren gehen können (Streaming / VoIP).