openHPI-Kurs: Wie funktioniert das Internet?

Protokolle der Transportschicht

Prof. Dr. Christoph Meinel
Hasso-Plattner-Institut
Universität Potsdam

Protokolle der Transportschicht (1/4)

Transportschicht im TCP/IP Schichtenmodell:

TCP/IP-Stack

4	Verarbeitung (Application Layer)
3	**Transport (Transport Layer)**
2	Vermittlung (Internet Layer)
1	Sicherung (Link Layer)
0	Bitübertragung (Hardware Layer)

Protokolle der Transportschicht (2/4)

- Protokolle der Transportschicht setzen auf dem verbindungslosen Datagrammdienst IP der Internetschicht auf
- Prominentestes Protokoll ist **TCP** – **Transmission Control Protocol**
- TCP leistet dabei scheinbar Unmögliches:
  - Auf Basis des unzuverlässigen IP-Datagrammdienstes bietet TCP einen zuverlässigen, gesicherten Transportdienst zum Datenaustausch zwischen zwei Endsystemen
- Grundidee:
  - TCP gleicht Fehler des IP-Datagrammdienstes aus …
- Erfolg des Internets ist neben IP vor allem dem glücklichen Design des TCP-Protokolls zu verdanken → **TCP/IP-Protokollsuite**

Protokolle der Transportschicht (3/4)

In der TCP/IP-Protokollsuite gibt es zwei grundlegende Transportprotokolle:

- **TCP – Transmission Control Protocol**:
  - bietet gesicherten, verbindungsorientierten Vollduplex-Kommunikationsdienst im Internet
  - ist sehr komplex und unterliegt steter Weiterentwicklung
- **UDP – User Datagram Protocol**:
  - bietet sehr einfachen, ungesicherten, verbindungslosen Kommunikationsdienst im Internet
  - aufgrund seiner Einfachheit bisher kaum Veränderungen unterworfen

Protokolle der Transportschicht (4/4)

Weitere Transportprotokolle der TCP/IP-Protokollfamilie setzen ihrerseits auf TCP oder UDP auf:

- ISO/T1 bis ISO/T4
- NetBIOS
- Real Time Protocol – RTP
- System Network Architecture – SNA
- …

openHPI-Kurs: Wie funktioniert das Internet?

Virtuelle Verbindungen mit TCP

Prof. Dr. Christoph Meinel
Hasso-Plattner-Institut
Universität Potsdam

Virtuelle Verbindung mit TCP (1/5)

In Rechnernetzen und Zusammenschlüssen gibt es zwei Typen von Diensten:

Verbindungslose Dienste:

- Kommunikation erfolgt ohne vorherigen Verbindungsaufbau
- Pakete werden einfach in das Netzwerk abgeschickt

→ Beispiel: Datagrammdienste IPv4, IPv6

Verbindungsorientierte Dienste:

- Kommunikation erfolgt erst nach Aufbau einer Verbindung
- alle Datenpakete werden demzufolge in korrekter Reihenfolge versendet und empfangen

→ Beispiel: TCP

Virtuelle Verbindung mit TCP (2/5)

TCP bietet **verbindungsorientierten Dienst**

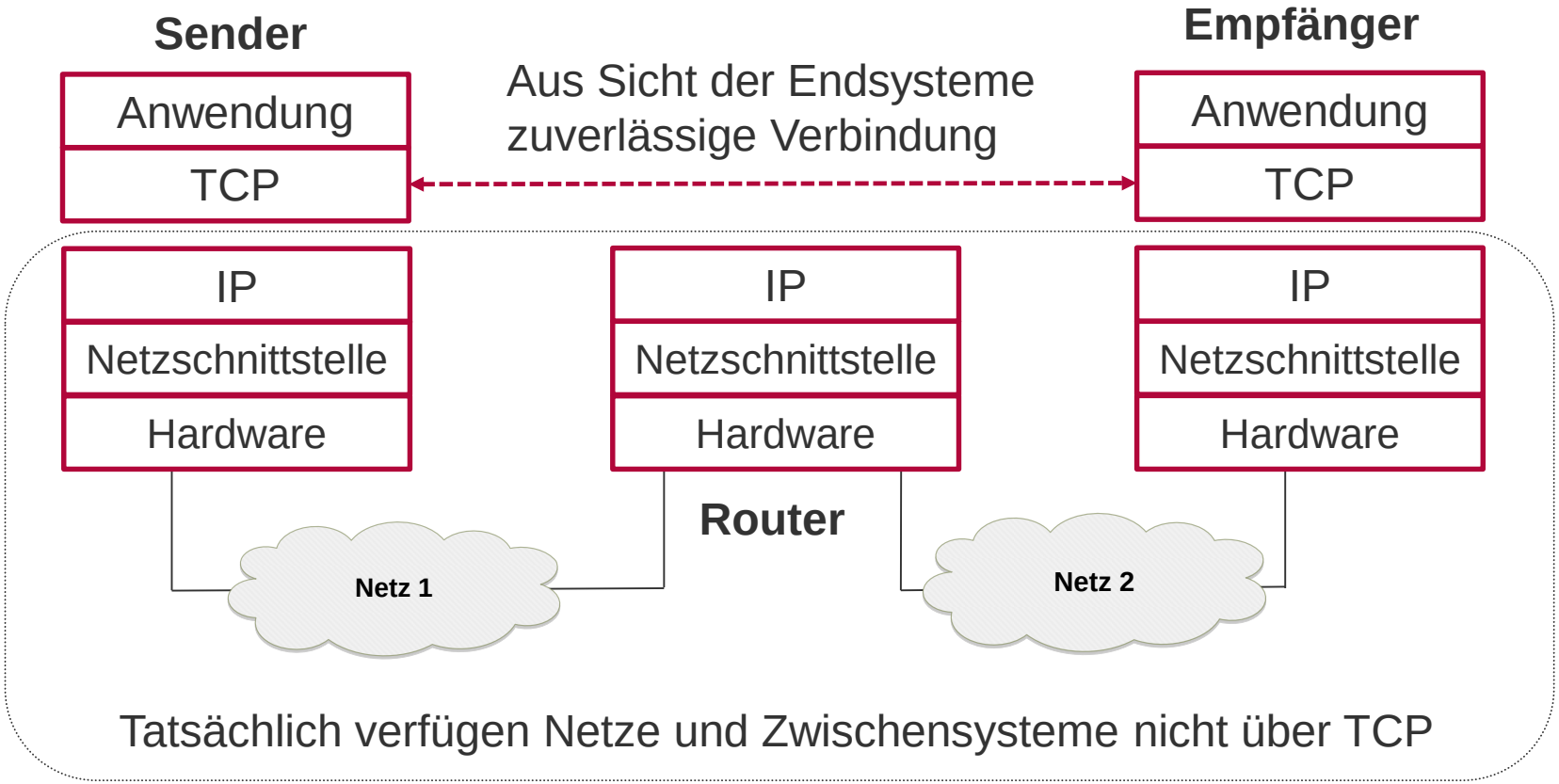
- TCP stellt aber lediglich **virtuelle Verbindung** bereit, d.h.
  - TCP-Prozesse auf den Endsystemen täuschen Verbindung softwaretechnisch vor
  - Wie schafft TCP die Illusion der Verbindung?
    - Mit Hilfe sogenannter **Sequenznummern** wird jeweils jede von A nach B und jede von B nach A gesendete TCP-Nachricht durchnummeriert
    - Anhand der Sequenznummern kann TCP beim Empfänger Vollständigkeit, richtige Reihenfolge, Duplikatfreiheit, … der gesendeten TCP-Nachrichten rekonstruieren

Virtuelle Verbindung mit TCP (3/5)

TCP bietet **verbindungsorientierten Dienst**

- Vor eigentlicher Datenübertragung baut TCP virtuelle Verbindung auf zum designierten Empfänger, die nach vollzogener Daten-Übertragung wieder abgebaut wird
  - Hierbei müssen insbesondere in beide Richtungen die Sequenznummern vereinbart und bestätigt werden
- TCP muss nur in den Endsystemen implementiert sein, nicht in den Zwischensystemen
  - TCP nutzt IP-Datagrammdienst zur Datenübertragung
  - jede TCP-Nachricht wird gekapselt in IP-Datagrammen über das Internet übertragen
- IP behandelt TCP-Nachrichten als reine Nutzdaten

Virtuelle Verbindung mit TCP (3/5)



Virtuelle Verbindung mit TCP (4/5)

Ende-zu-Ende-Übertragung

TCP erlaubt ausschließlich Datenübertragungen zwischen **zwei** dezidierten Endsystemen

- Verbindung verläuft von einer Anwendung beim Sender zu einer Anwendung beim Empfänger
- Aufbau der virtuellen Verbindung ist allein Sache der beiden Endsysteme, die zu überbrückenden Netze und Zwischensysteme leiten Nachrichten nur in Form von IP-Datagrammen weiter
- Multicast oder Broadcast ist mit TCP nicht möglich

Vollduplex Übertragung mit TCP

Vollduplex Übertragung

- TCP gestattet bidirektionale Ende-zu-Ende Datenübertragung
- Jeweils ausgestattet mit Eingangspuffern können Sender und Empfänger gleichzeitig senden

Zuverlässige Übertragung mit TCP

TCP garantiert

- fehlerfreie Datenübertragung,
- kein Datenverlust,
- keine vertauschten Datenpakete,
- ...

TCP bedient sich dazu der folgenden Techniken

- Retransmission – Neuübertragung
- Adaptive Neuübertragung
- Überlastkontrolle – Congestion Control

openHPI-Kurs: Wie funktioniert das Internet?

Retransmission bei TCP

Prof. Dr. Christoph Meinel
Hasso-Plattner-Institut
Universität Potsdam

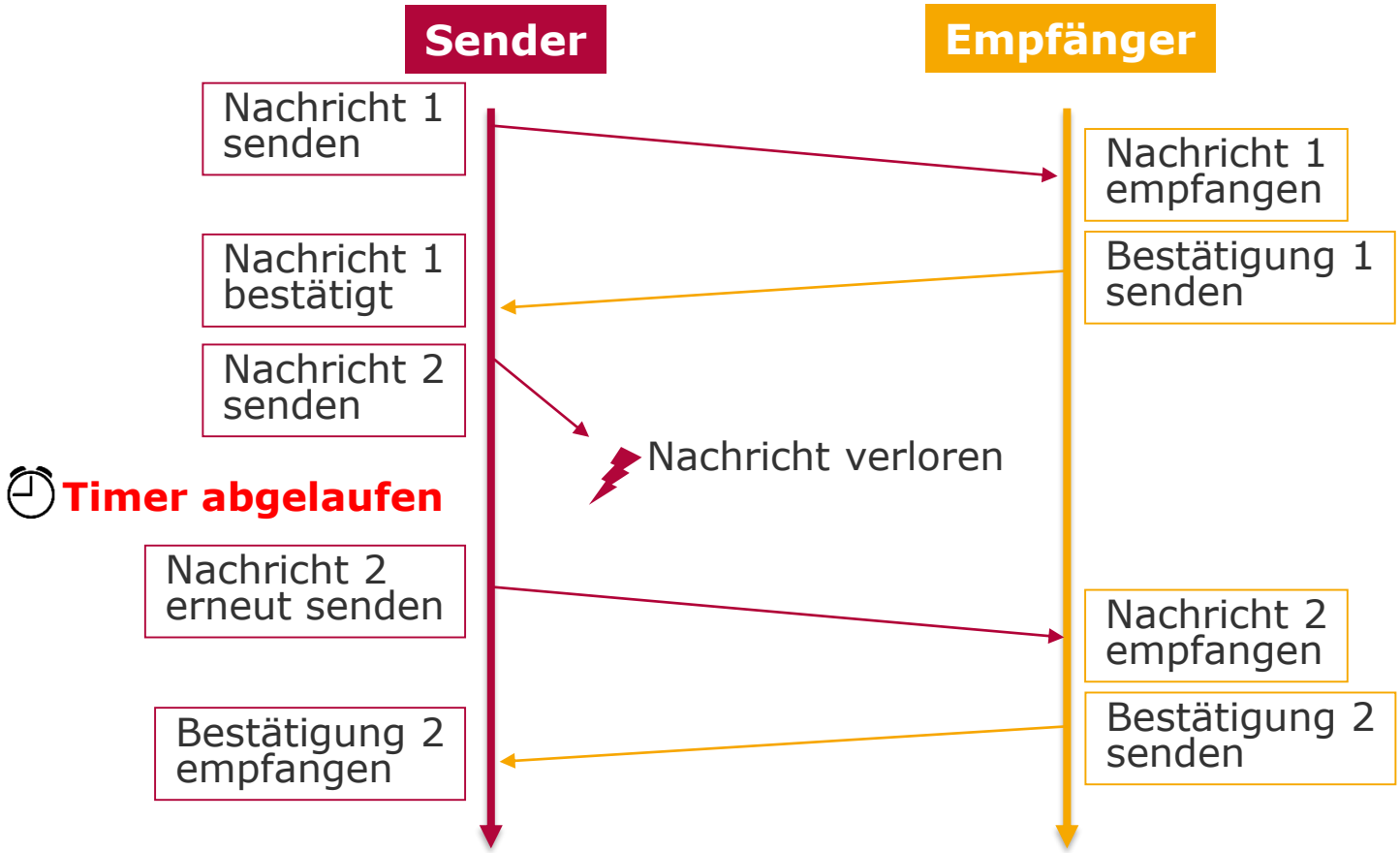
Retransmission (1/4)

Die von TCP zur Korrektur im Fehlerfall initiierte Neuübertragung von Datenpaketen ist Grundtechnik zur Gewährleistung fehlerfreier Übertragung – **Retransmission**

- Gesendete TCP-Nachrichten werden dem Sender nach Empfang vom Empfänger bestätigt – **Acknowledgement**
- Woher weiß aber Empfänger überhaupt, dass eine TCP-Nachricht gesendet wurde?
- Sender startet vor Übertragung eines Datenpakets eine Uhr – **Timer**:
  - Läuft Timer vor Eintreffen einer Empfangsbestätigung ab, gilt Datenpaket als verloren und wird neu übertragen
- Um Neuübertragung zu erzwingen, sendet Empfänger im Fall beschädigter Nachrichten kein Acknowledgement und wartet einfach auf automatische Retransmission nach Timer-Ablauf

Retransmission (2/4)

TCP–Retransmission bei Zeitüberschreitung



Retransmission (3/4)

- Wahl der Zeitschranke für Timer bestimmt Effizienz des Datendurchsatzes im Internet
- Innerhalb eines LANs sind kürzere Zeiten sinnvoller als bei Weitverkehrsverbindungen:
  - Erfolgt Neuübertragung …
    - zu früh, wird Internet mit überflüssigen Duplikaten überflutet → Durchsatz sinkt
    - zu spät, müssen viele Nachrichten zwischengespeichert werden → Überlauf der Warteschlangen

Retransmission (4/4)

- Fixe Timer-Zeitschranke kann nie allen Situationen gerecht werden
- TCP bietet deshalb komplexen adaptiven Mechanismus zur Anpassung der Zeitschranke: **Adaptive Retransmission**
- TCP überwacht dazu Netzlast für jede Verbindung vermittels der Paketumlaufzeit – **Round-Trip-Time** – Zeitspanne zwischen Aussendung eines Pakets und Eintreffen dessen Empfangsbestätigung:
  - Messung wird für jede Sendung ausgeführt und mit vorher-gehenden Messungen für gleiche Verbindung verglichen
  - Über gleitenden Mittelwert – **Smoothed Round-Trip-Time** – wird Zeitspanne adaptiv an momentane Lastsituation angepasst

openHPI-Kurs: Wie funktioniert das Internet?

TCP-Flusskontrolle

Prof. Dr. Christoph Meinel
Hasso-Plattner-Institut
Universität Potsdam

Flusskontrolle – Sliding Window Protocol (1/4)

TCP regelt Datenfluss über eine Verbindung mit Hilfe eines Schiebefenster-Protokolls – **Sliding Window Protocol**

- Schiebefenster-Protokoll arbeitet mit adaptiven, lastabhängigen Fenstern – **Windows**, wobei Quittierung und Zuweisung eines Fensters voneinander entkoppelt sind

Beispiel:

Nachricht der Länge 2.500 Byte wird per TCP-Verbindung mit Fenstergröße 1.500 Byte (F = 1.500) von A nach B übertragen

- A sendet erste 1.000 Byte
- B nimmt diese entgegen, schreibt sie in Eingangspuffer, setzt Fenstergröße auf 1.500-1.000 = 500 (F=500) und bestätigt Annahme der ersten 1.000 Byte (ACK=1.000) und Fenstergröße (F=500)

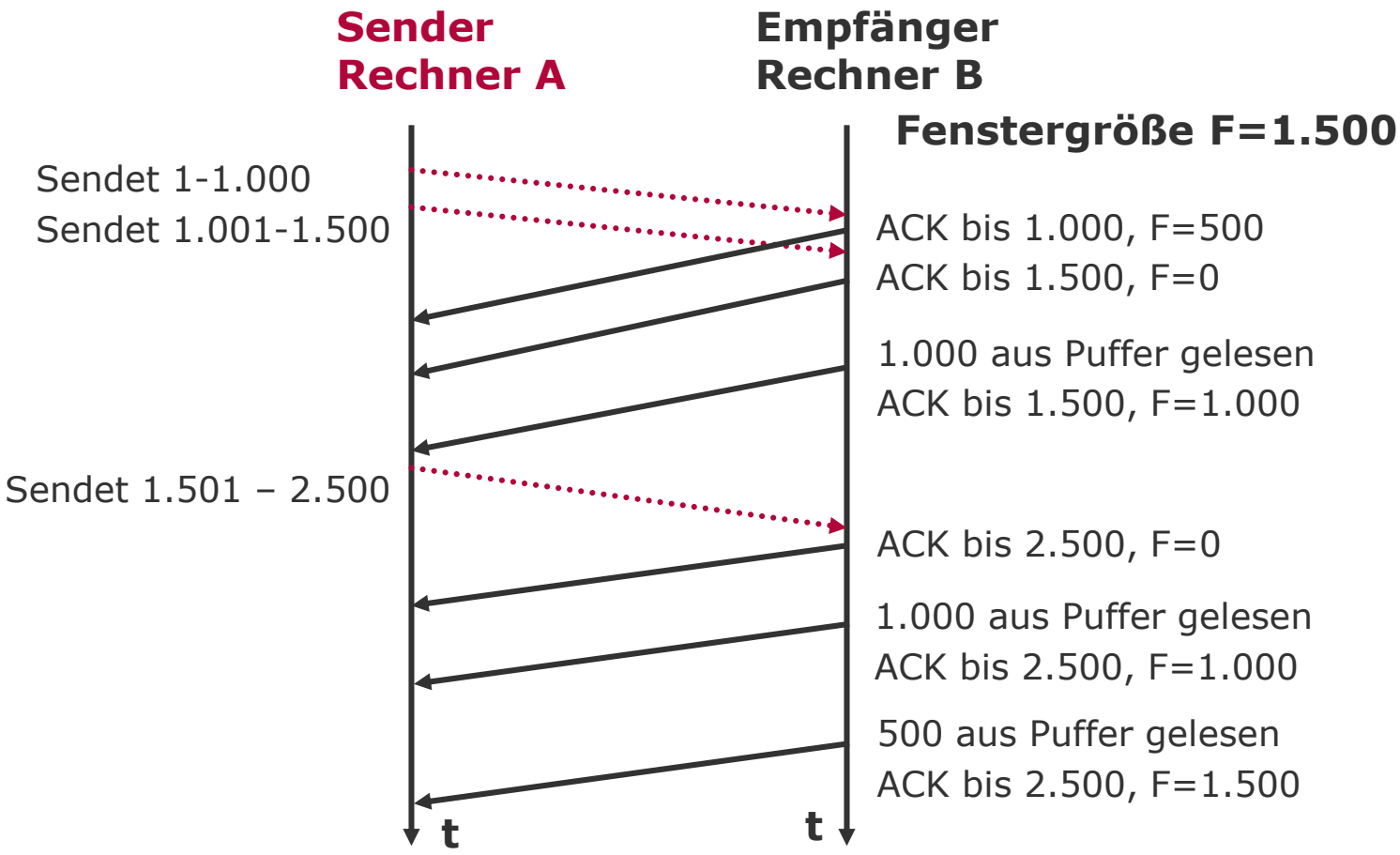
Flusskontrolle – Sliding Window Protocol (2/4)

Beispiel:

Nachricht der Länge 2.500 Byte wird per TCP-Verbindung mit Fenstergröße 1.500 Byte von A nach B übertragen

- …
- A überträgt gemäß erlaubter Fenstergröße nächste 500 Byte
- B nimmt diese entgegen, setzt Fenstergröße auf 0 und bestätigt Empfang und neue Fenstergröße
- A muss nun warten bis Anwendung auf B aus Eingangspuffer 1.000 Byte ausgelesen hat, Fenstergröße wieder auf 1.000 Byte gesetzt und entsprechende Mitteilung geschickt (ACK=1.500, F=1.000) ist
- A kann nun den Rest der Nachricht (1.000 Byte) schicken

Flusskontrolle – Sliding Window Protocol (3/4)



Flusskontrolle – Sliding Window Protocol (4/4)

Silly Window Syndrom

- Ohne zusätzliche Vorkehrungen würde sich bei schnellen Verbindungen die Fenstergröße nahe 0 einpegeln und die Übertragungsstrecke nur unzureichend genutzt
- Empfänger gibt deshalb Rückmeldung über verfügbare Fenstergröße erst dann, wenn wieder mindestens 50% der maximalen Fenstergröße zur Verfügung stehen
- Sender sorgt dafür, dass die verfügbare Fenstergröße nicht vollständig ausgenutzt wird

openHPI-Kurs: Wie funktioniert das Internet?

TCP-Überlastkontrolle

Prof. Dr. Christoph Meinel
Hasso-Plattner-Institut
Universität Potsdam

Überlastkontrolle – Congestion Control (1/5)

Überlaststeuerung – **Congestion Control** – ist eines der schwierigsten Probleme für TCP, da TCP Überlastsituationen nicht direkt erkennen kann:

- Datenpakete werden per IP auf unterschiedlichen Wegen durch das Internet befördert, sodass keine direkten Rückschlüsse auf Überlast bestimmter Zwischensysteme gezogen werden können
- Verschiedene TCP-Instanzen können nicht miteinander kooperieren

Idee: TCP interpretiert Zahl der verlorenen (also nicht bestätigten) Segmente als Maß für Überlast und Parameter für Bestimmung des Congestion Windows zur Regelung der Überlast

Überlastkontrolle – Congestion Control (2/5)

Slow-Start Algorithmus – zur schnellen Anpassung des Congestion Windows:

- Verbindungen werden mit kleiner Fenstergröße gestartet, die dann solange exponentiell gesteigert wird, bis Pakete verloren gehen…

Congestion-Avoidance Algorithmus – zum Abbau von Überlastsituationen:

- Häuft sich der Segmentverlust – d.h. Quittungen erreichen Sender nicht innerhalb der festgesetzten Zeitspanne – senkt TCP die Sende-Datenrate, um eine Überlastung des Netzwerkes nicht mit ständiger Wiederholung von Sendungen zu verschärfen

Überlastkontrolle – Congestion Control (3/5)

Nagle-Algorithmus (RFC 896 wurde 2016 durch RFC 7805 ersetzt)

Einschränkung des Overheads beim Senden:

- Sind TCP-Segmente nur von kurzer Länge, machen Steuer- und Kontrollinformationen im Header den Großteil der zu übertragenden Daten aus
- Quittierung eines Segments wird deshalb solange als möglich verzögert, um diese z.B. mit weiteren Nutzdaten zu senden
- Sendung eigener Nutzdaten wird solange wie möglich verzögert, um diese erst zusammen mit ausstehenden Quittierungen zu versenden
- Der Nagle-Algorithmus sammelt alle Nutzdaten bis zur nächstfälligen Quittung und überträgt diese dann gemeinsam

Überlastkontrolle – Congestion Control (4/5)

Bestimmung der Round Trip Time (RTT) (1/2):

- **Originalimplementierung**: RTT wird für jedes gesendete Segment ermittelt und ein gewichtetes Mittel errechnet
- **Karn/Partridge-Algorithmus**:
  - Bei Originalimplementierung kann tatsächlicher Wert unterschätzt werden, da fälschlich auch Zeit zwischen Neuaussendung eines verlorenen geglaubten Pakets und verspätet eintreffender Quittierung des Segments gemessen werden kann
  - Karn/Partridge-Algorithmus zur RTT-Berechnung basiert nur auf Messungen bei tatsächlich vom Empfänger quittierten Segmenten
  - Zusätzlich wird Zeitschranke nach jedem erfolgreichen Versand angehoben

Überlastkontrolle – Congestion Control (5/5)

Bestimmung der Round Trip Time (RTT) (2/2):

- **Karel/Jacobson Algorithmus**
  - Verfeinerung des Karn/Partridge-Algorithmus durch Gewichtung der Schwankungen der gemessenen Netzumlaufzeit
  - Zusätzlich wird Berechnung des Timeouts angepasst

openHPI-Kurs: Wie funktioniert das Internet?

TCP Ports

Prof. Dr. Christoph Meinel
Hasso-Plattner-Institut
Universität Potsdam

TCP Ports (1/4)

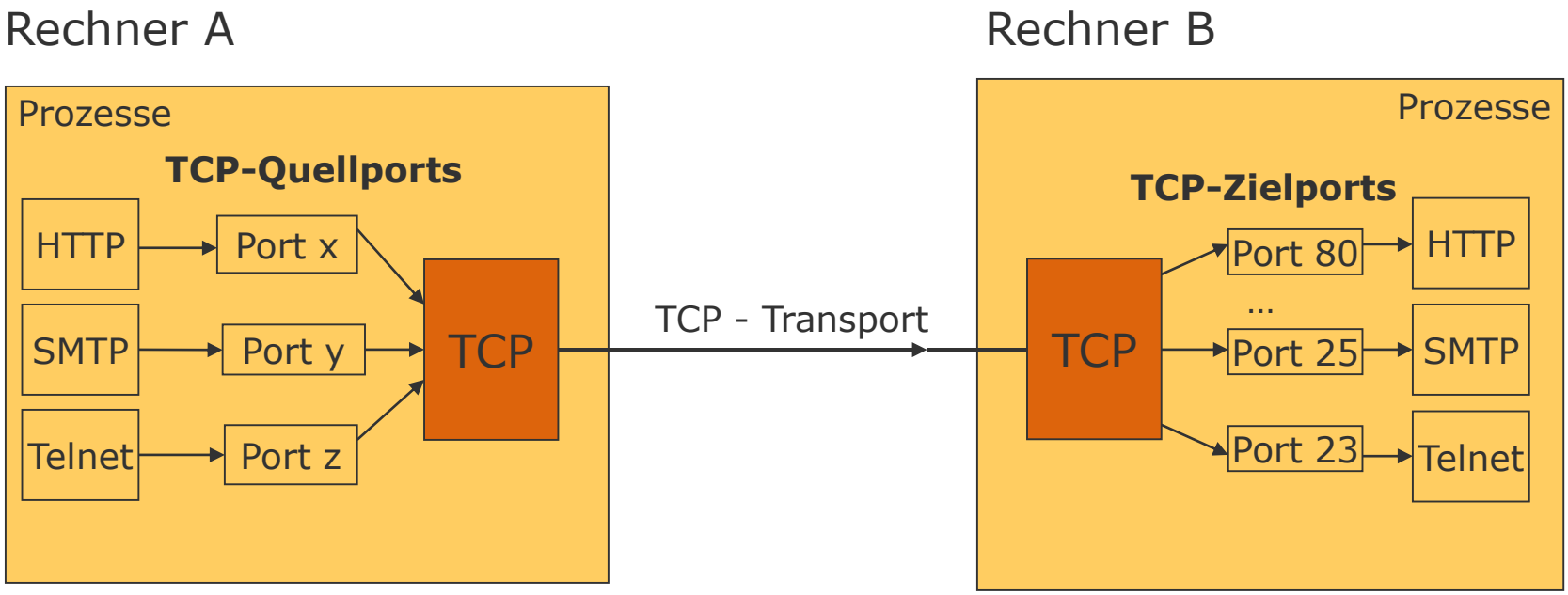
- Um TCP-Verbindung aufrechterhalten zu können, müssen Sender und Empfänger jeweils Endpunkte – **Sockets** – für die Kommunikation einrichten
- Jeder Socket verfügt über reservierten Speicherplatz zum Puffern der empfangenen oder zu sendenden Daten
- Zuordnung der Sockets ist auf beide Kommunikationspartner und Zeitdauer der jeweiligen TCP-Verbindung beschränkt
- Jeder Socket besitzt **Socket-Nummer**, die sich aus
  - IP-Adresse des Rechners und
  - lokal zuordenbarer 16-Bit-Nummer – **Port** – ergibt
- Ports sind **Service Access Points** der Transportschicht

TCP Ports (2/4)

- Ports mit Nummern
  - 0 – 1.023 – **Well Known Ports** – sind weltweit eindeutig für Standarddienste reserviert, z.B.
    - 80 – HTTP, 23 – Telnet, 53 – DNS, …
  - 1.024 – 65.535 können beliebigen Anwendungsprogrammen zugeordnet werden
- Man unterscheidet:
  - **Reservierte Ports**:
    - 0 – 255 für TCP/IP-Anwendungen
    - 256 – 1.023 für bestimme UNIX-Anwendungen
  - **Registrierte Ports**:
    - 1.024 – 49.151 von IANA registriert
  - **Private, dynamische Ports**: Rest

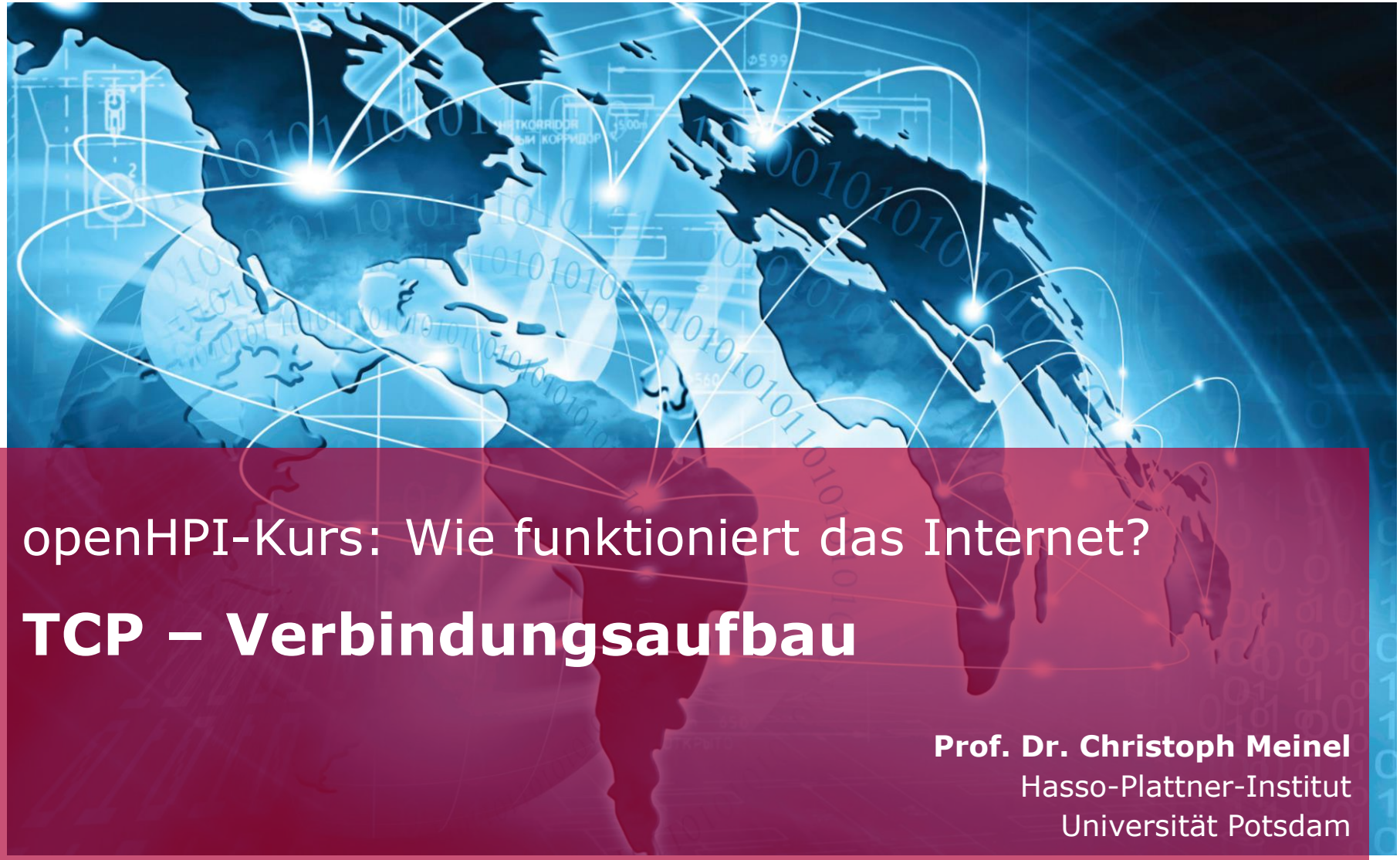
TCP Ports (3/4)

TCP–Verbindungen zwischen verschiedenen Anwendungsprozessen verlaufen über TCP-Ports



TCP Ports (4/4)

- Portnummern sind zu wählen beim Verbindungsaufbau zwischen zwei Anwendungsprogrammen:
  - **Zielport**: Portnummer der gewünschten Anwendung
  - **Quellport**: Nicht reservierte und zur Zeit nicht anders genutzte Portnummer
- TCP-Verbindung kann anhand der beiden Socket-Nummern über das Tupel (Socket Sender, Socket Empfänger) eindeutig identifiziert werden
- Ein Server-Socket kann gleichzeitig mehreren Verbindungen dienen
- Anwendungsprogramme nutzen den TCP-Transportdienst über sogenannte **TCP-Primitive** – Primitives, z.B. Request, Response, Confirm, Read, Write, …, die über den Socket implementiert sind



openHPI-Kurs: Wie funktioniert das Internet?

TCP – Verbindungsaufbau

Prof. Dr. Christoph Meinel
Hasso-Plattner-Institut
Universität Potsdam

Zuverlässiger Verbindungsauf- und Abbau (1/5)

- TCP verlangt, dass Sender und Empfänger vor ihrer Datenübertragung eine Verbindung aufbauen
  - eventuelle Nachrichtenduplikate aus früheren Verbindungen können so ignoriert werden
- Wollen Kommunikationspartner die Verbindung beenden, sorgt TCP vor Abbau der Verbindung dafür, dass alle gesendeten, aber noch nicht angekommenen TCP-Nachrichten zugestellt werden
- Zum Auf- bzw. Abbau von TCP-Verbindungen werden drei bzw. vier Nachrichten ausgetauscht → **Drei-Wege-Handshake**

Zuverlässiger Verbindungsauf- und Abbau (2/5)

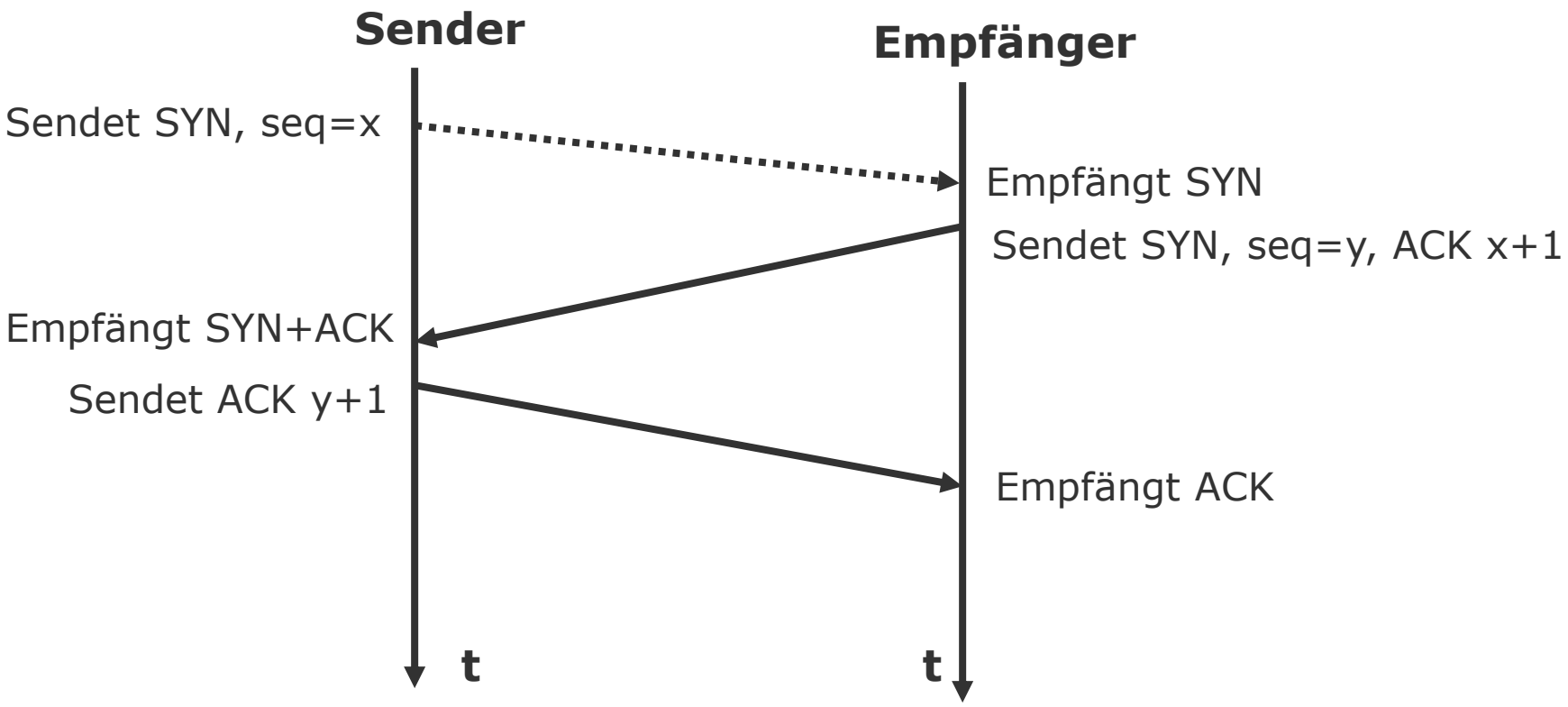
Drei-Wege-Handshake sorgt dafür, dass beide Seiten zum Datenempfang bereit sind und die zur Identifikation der Verbindung essentiellen → **Sequenznummern** ausgetauscht werden

Drei-Wege-Handshake zum Verbindungsaufbau:

- Initiator sendet spezielles Segment - SYN-Segment - , das eine Sequenznummer x enthält und dessen Synchronisationsbit auf 1 gesetzt ist
- Empfänger des SYN-Segments bestätigt die im SYN-Segment enthaltene Sequenznummer x+1, lässt Synchronisationsbit gesetzt und sendet eigene Sequenznummer y
- Initiator bestätigt Empfang der Synchronisations-Antwort mit Sequenznummer y+1

Zuverlässiger Verbindungsauf- und Abbau (3/5)

Verbindungsaufbau mit dem Drei–Wege–Handshake:



Zuverlässiger Verbindungsauf- und Abbau (4/5)

Zum zuverlässigen beidseitigen Abbau der Verbindung wird **modifiziertes Drei-Wege-Handshake** (eigentlich Vier-Wege-Handshake) durchgeführt

Drei-Wege-Handshake zum Verbindungsabbau (1/2):

- Initiator (Sender) sendet spezielles Ende-Segment – FIN-Segment, dessen FIN-Bit auf 1 gesetzt ist und das eine End-Sequenznummer x enthält
- Gegenseite bestätigt übermittelte End-Sequenznummer und nimmt für diese Verbindung keine weiteren Segmente mehr an. TCP-Instanz benachrichtigt lokale Anwendung über Verbindungsabbau
- …

Zuverlässiger Verbindungsauf- und Abbau (5/5)

Drei-Wege-Handshake zum Verbindungsabbau (2/2):

- …
- Informierte Anwendung startet ihrerseits Verbindungsabbau durch Versand eines FIN-Segments mit eigener End-Sequenznummer y und erneuter Bestätigung der zuletzt empfangen Sequenznummer
- Sender bestätigt Empfang und die übermittelte End-Sequenznummer

openHPI-Kurs: Wie funktioniert das Internet?

TCP-Segmente

Prof. Dr. Christoph Meinel
Hasso-Plattner-Institut
Universität Potsdam

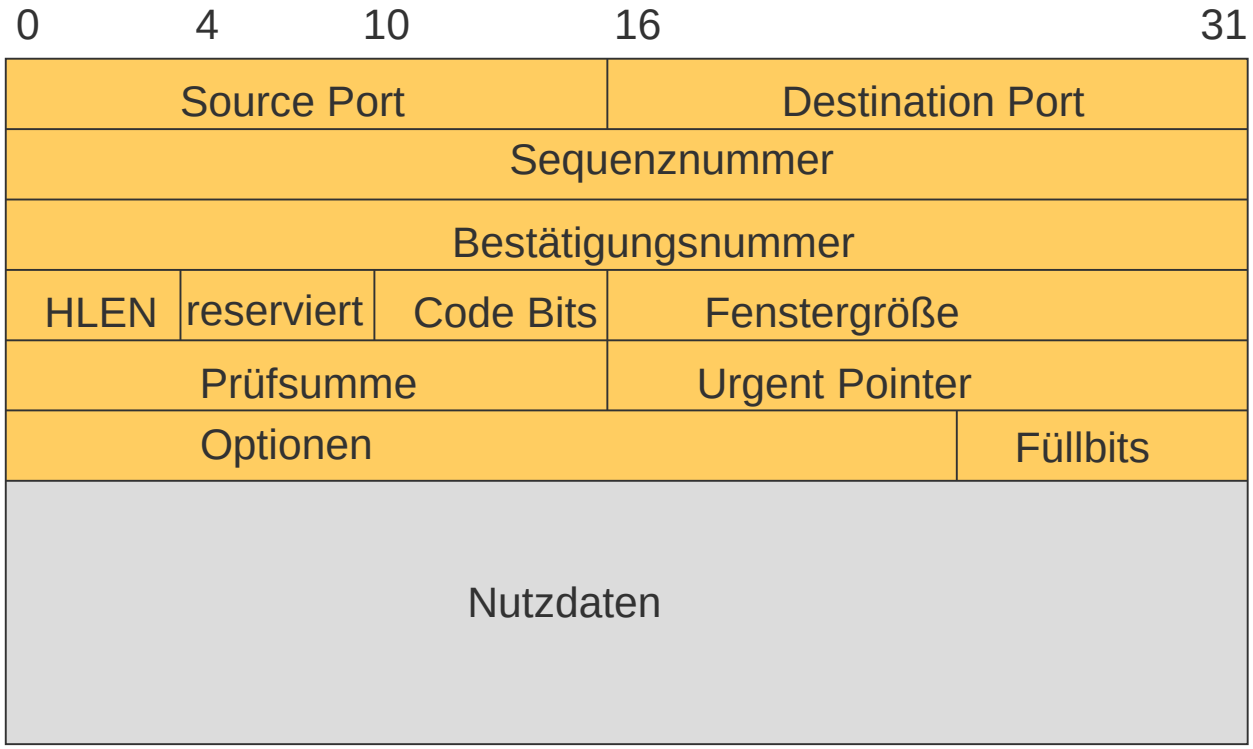
TCP-Segmente (1/5)

TCP überträgt Nachrichten in Form von einzelnen Datenblöcken – **TCP-Segmente**

- Bei Aufteilung der Nachricht in Segmente – **Segmentierung** – wird jedes Segment mit eigenem TCP-Header versehen, der alle notwendigen Steuer- und Kontrollinformationen enthält
- Maximale Segmentgröße wird jeweils beim Verbindungsaufbau vereinbart
- Zum Transport werden TCP-Segmente in IP-Datagrammen gekapselt und mit dem IP-Datagrammdienst gesendet
- Geht IP-Datagramm mit TCP-Segment verloren, wird das TCP-Segment nicht bestätigt und nach Timeout neu übertragen

TCP-Segmente (2/5)

Format eines TCP-Segments



TCP-Segmente (3/5)

Aufbau des TCP-Headers (1/3):

- **Source Port** (16 Bit) – Quell-Port, der mit Anwendungsprozess assoziiert ist, der die Verbindung gestartet hat
- **Destination Port** (16 Bit) – Ziel-Port, ist mit Anwendungsprozess assoziiert, der die übertragenen Daten entgegennimmt
- **Sequenznummer** (32 Bit)
  - Zur Durchnummerierung der in Senderichtung übermittelten Segmente
  - Initialer Wert wird beim Verbindungsaufbau ausgetauscht
  - Erhöhung erfolgt jeweils um Anzahl der bereits gesendeten Bytes
- …

TCP-Segmente (4/5)

Aufbau des TCP-Headers (2/3):

- **Bestätigungsnummer** (32 Bit) – dient auf Empfängerseite zur Bestätigung empfangener Segmente
- **Header Length** (4 Bit) – Länge des TCP-Headers in 32-Bit Worten
- **Code Bits** (6 Bit) – zeigen an, welche Felder im Header gültig sind
- **Sendefenster** – **Window** (16 Bit) – Zahl der Bytes, die Empfänger in seinen Eingabepuffer aufnehmen kann
- **Prüfsumme** (16 Bit) – berechnet aus TCP-Header, den TCP-Nutzdaten und einem Pseudo-Header (der aus TCP- und IP-Daten berechnet wird)
- …

TCP-Segmente (5/5)

Aufbau des TCP-Headers (3/3):

- **Urgent Pointer** (16 Bit) – zeigt an, dass neben den Nutzdaten dringliche – Out-of-Band – Daten, z.B. Interrupts, übertragen werden, die so schnell wie möglich, z.B. ohne Ablage im Eingangspuffer, an Anwendungsprozess übermittelt werden sollen
- **Optionen** – zur Einbeziehung zusätzlicher Funktionalität, z.B. Maximum Segment Size – MSS, Window Scale Option – WSopt, Timestamps Option – TSopt, Selective Acknowledgement, Connection Count – CC
- **Füllbits** – füllen Länge des TCP-Segment-Headers auf Wortgrenze von 32 Bit auf

openHPI-Kurs: Wie funktioniert das Internet?

Transportprotokoll UDP

Prof. Dr. Christoph Meinel
Hasso-Plattner-Institut
Universität Potsdam

UDP – User Datagram Protocol (1/4)

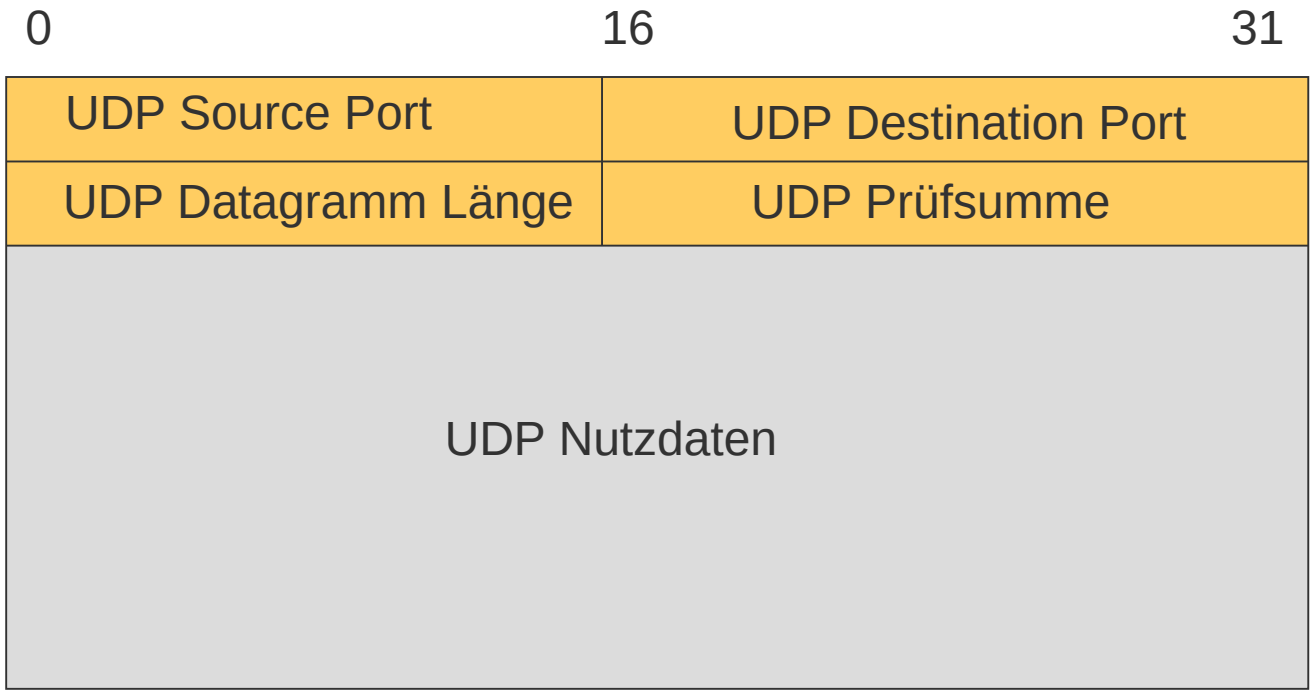
- TCP stellt zuverlässigen, verbindungsorientierten Transportdienst zur Verfügung, Implementierung und Abarbeitung des TCP-Protokolls erfordern jedoch erheblichen Aufwand, z.B. für Verbindungsauf- und abbau
- Als Alternative und zum Versand kurzer Nachrichten ohne vorherigen Verbindungsaufbau auf der Transportebene dient **UDP – User Datagram Protocol**
- UDP bietet einen verbindungslosen, unzuverlässigen Transportdienst
- UDP setzt (wie TCP) direkt auf IP auf und bietet als zusätzliche Funktionalität lediglich die Bereitstellung von Kommunikationsports – **UDP-Ports** – zur Verknüpfung der kommunizierenden Anwendungen

UDP – User Datagram Protocol (2/4)

- UDP wird von verschiedenen Protokollen der Anwendungsschicht genutzt, die sich auf **einfache Frage/Antwort-Interaktionen** beschränken, z.B.
  - TFTP - 69, DNS - 53, NTP - 123, RPC - 111, LDAP - 389, …
- UDP nimmt Daten von Anwendung über deren Ports entgegen und gibt sie als UDP-Datagramme gekapselt an IP zur Übertragung weiter – **Portmultiplexing**
- Für TCP und UDP festgelegte Portnummern können sich unterscheiden, müssen aber übereinstimmen, wenn sie von beiden Protokollen genutzt werden

UDP – User Datagram Protocol (3/4)

Aufbau eines UDP-Datenpakets – UDP-Datagramm:



UDP – User Datagram Protocol (4/4)

Aufbau des Headers eines UDP-Datagramms:

- **Source Port** (16 Bit) – Portnummer des Absenders
- **Destination Port** (16 Bit) – Portnummer des Empfängers
- **Länge** (16 Bit) – Länge des UDP-Datagramms in Byte, Minimum 8 Byte (Headerlänge)
- **Prüfsumme** (16 Bit) – über UDP-Header, UDP-Nutzdaten und Pseudoheader mit dem bei IP verwendeten Algorithmus (Berechnung ist optional)

UDP – Anwendungsfälle (1/2)

UDP wird genutzt von bekannten Diensten, wie z.B.

- DHCP – Dynamic Host Configuration Protocol
- DNS – Domain Name System
- NTP – Network Time Protocol
- ...

UDP kann genutzt werden, wenn

- Paketverluste tragbar sind oder
- die Anwendung die Paketverluste selbst handhabt
- keine Segmentierung notwendig ist
- sich Verbindungsaufbau nicht lohnt, z.B. bei DNS etc.
  - geht Paket verloren, erfolgt einfach neue Anfrage...
- ...

UDP – Anwendungsfälle (2/2)

Weiterhin kann UDP für Echtzeitdaten genutzt werden, z.B.

- Datenübertragung mit **RTP** (Real-Time Protocol)
- „Verbindungsaufbau" mit **SIP** (Session Initiation Protocol)
- Anwendungen wie Streaming von Video, VoIP etc.

Für verschlüsselte Übertragungen wird allerdings meist das verbindungsorientierte TCP bevorzugt, weil sonst jedes Paket einzeln verschlüsselt werden müsste (Overhead)

- Es gibt Überlegungen und Versuche, UDP zu erweitern und sicher zu machen, z.B. **SRTP** (RFC 3711) oder **DTLS** (RFC 6347)

openHPI-Kurs: Wie funktioniert das Internet?

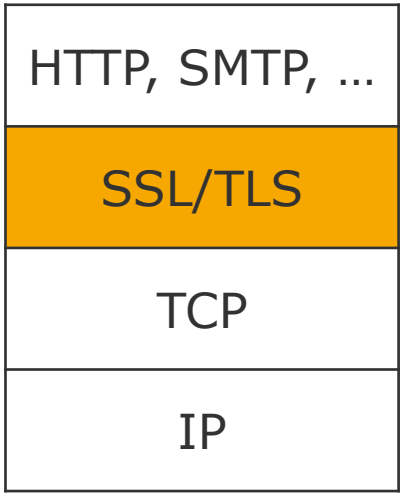
Absicherung auf der Transportschicht

Prof. Dr. Christoph Meinel
Hasso-Plattner-Institut
Universität Potsdam

Absicherung der Transportschicht im TCP/IP-Protokollstapel

- Um bestimmte Sicherheitsziele und –anforderungen für verschiedene Internetanwendungen zu erreichen braucht es geeignete Sicherheitsarchitekturen
- **Absicherung der Protokolle** – also die Bereitstellung von Sicherheitsmechanismen zur Gewährleistung der Sicherheitsziele – kann im TCP/IP-Protokollstapel auf unterschiedlichen Schichten erfolgen, z.B. auf der **Transportschicht**
- Geschickter Einsatz von Sicherheitsmechanismen auf der Transportschicht erübrigt
  - Absicherung von der vielen Protokolle auf der Anwendungsschicht
  - aufwändig Absicherung auf Internetschicht

→ **SSL/TLS Protokolle**



SSL / TLS

SSL/TLS – Transport Layer Security (1/4)

- Zum vertraulichen Datentransport über das WWW entwickelte Netscape 1994 eine eigene sichere Transportinfrastruktur:
  - HTTP über SSL – **HTTPS**
- Idee: Mit **SSL – Secure Socket Layer** – wird zwischen Anwendungsschicht (HTTP) und Transportschicht (TCP) eine neue Protokollschicht zum vertraulichen Transport im TCP/IP-Protokollstapel integriert
- Auch andere Protokolle der Anwendungsschicht des TCP/IP-Protokollstapels können ohne weitere Anpassungen die durch SSL bereitgestellten sicheren Verbindungen nutzen, z.B.
  - Online-Banking, E-Shopping, …
- SSL wurde in der Version 3.0 von IETF als **TLS – Transport Layer Security** standardisiert. Sprechen darum im Folgenden von **SSL/TLS**

SSL/TLS – Transport Layer Security (2/4)

- TLS verfolgt dieselben Ziele wie SSL:
  - kryptographische Sicherheit
  - Interoperabilität
  - Erweiterbarkeit
  - Effizienz
- TLS häufig eingesetzt in der Absicherung von Email-Verbindungen (IMAP+TLS, SMTP+TLS), z.B. bei GoogleMail

SSL/TLS – Transport Layer Security (3/4)

SSL/TLS bietet Kommunikationspartnern:

- **Private Verbindung** – nach anfänglichem Handshake-Verfahren zum sicheren Schlüsselaustausch werden Daten symmetrisch verschlüsselt
- **Authentifikation** über asymmetrische Verschlüsselungsverfahren
- **Zuverlässige Verbindung** – Nachrichtentransport überprüft Unversehrtheit der transportierten Daten über Message Authentication Code – MAC

SSL/TLS – Transport Layer Security (4/4)

SSL/TLS im TCP/IP-Schichtenmodell: