

# 11. Foliensatz

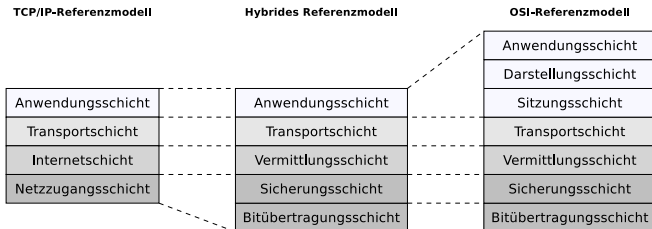
## Betriebssysteme und Rechnernetze

Prof. Dr. Christian Baun

Frankfurt University of Applied Sciences  
(1971–2014: Fachhochschule Frankfurt am Main)  
Fachbereich Informatik und Ingenieurwissenschaften  
[christianbaun@fb2.fra-uas.de](mailto:christianbaun@fb2.fra-uas.de)

# Transportschicht

- Aufgaben der Transportschicht (Transport Layer):
  - Enthält **Ende-zu-Ende-Protokolle** für Interprozesskommunikation
  - Adressierung der Prozesse mit **Portnummern**
  - Unterteilung der Daten der Anwendungsschicht in **Segmente**



- Geräte: Gateway
- Protokolle: TCP, UDP

# Sinnvolle Themen zur Transportschicht...

- ... und was aus Zeitgründen davon übrig bleibt...
  - Eigenschaften von Transportprotokollen
  - Adressierung in der Transportschicht
  - User Datagram Protocol (UDP)
    - Aufbau von UDP-Segmenten
    - Arbeitsweise
  - Transmission Control Protocol (TCP)
    - Aufbau von TCP-Segmenten (Dieser Teil wurde reduziert)
    - Arbeitsweise
    - ~~Flusskontrolle (Flow Control)~~
    - ~~Überlastkontrolle (Congestion Control)~~
    - Denial of Service-Attacken via SYN-Flood

# Herausforderungen für Transportprotokolle

- Das Protokoll IP auf der Vermittlungsschicht arbeitet **verbindungslos**
  - IP-Pakete werden unabhängig von anderen zum Ziel vermittelt (*geroutet*)
  - Vorteil: Geringer Overhead
- Nachteile aus Sicht der Transportschicht
  - IP-Pakete gehen **verloren** oder werden **verworfen**, weil TTL abgelaufen
  - IP-Pakete erreichen ihr Ziel häufig in der **falschen Reihenfolge**
  - **Mehrere Kopien** von IP-Paketen erreichen das Ziel
- Gründe:
  - Große Netze sind nicht statisch  $\implies$  ihre Infrastruktur ändert sich
  - Übertragungsmedien können ausfallen
  - Die Auslastung und damit die Verzögerung der Netze schwankt
- Diese Probleme sind in Computernetzen alltäglich
  - Je nach Anwendung müssen Transportprotokolle diese Nachteile ausgleichen

# Eigenschaften von Transportprotokollen

- Gewünschte Eigenschaften von Transportprotokollen sind u.a.
  - Garantierte Datenübertragung
  - Einhaltung der korrekten Reihenfolge der Daten
  - Unterstützung beliebig großer Datenübertragungen
  - Der Sender soll in der Lage sein, den Datenfluss, also die Übertragungsrate zu kontrollieren  $\implies$  Flusskontrolle (**leider gestrichen aus Zeitgründen**)
- Es sind also Transportprotokolle nötig, die die negativen Eigenschaften der Netze in die (positiven) Eigenschaften umwandeln, die von Transportprotokollen erwartet werden
- Die am häufigsten verwendeten Transportprotokolle:
  - **UDP**
  - **TCP**
- Adressierung erfolgt in der Transportschicht mit **Sockets**

# Adressierung in der Transportschicht

- Jede Anwendung, die TCP oder UDP nutzt, hat eine **Portnummer**
  - Diese gibt an, welcher Dienst angesprochen wird
  - Bei TCP und UDP ist die Portnummer 16 Bits groß
    - Portnummern liegen somit im Wertebereich 0 bis 65.535
- Portnummern können im Prinzip beliebig vergeben werden
  - Es gibt Konventionen, welche Standardanwendungen welche Ports nutzen

Portnummer	Dienst	Beschreibung
21	FTP	Dateitransfer
22	SSH	Verschlüsselte Terminalemulation (Secure Shell)
23	Telnet	Terminalemulation zur Fernsteuerung von Rechnern
25	SMTP	E-Mail-Versand
53	DNS	Auflösung von Domainnamen in IP-Adressen
67	DHCP	Zuweisung der Netzwerkkonfiguration an Clients
80	HTTP	Webserver
110	POP3	Client-Zugriff für E-Mail-Server
143	IMAP	Client-Zugriff für E-Mail-Server
443	HTTPS	Webserver (verschlüsselt)
993	IMAPS	Client-Zugriff für E-Mail-Server (verschlüsselt)
995	POP3S	Client-Zugriff für E-Mail-Server (verschlüsselt)

- Die Tabelle enthält nur eine kleine Auswahl bekannter Portnummern

## Ports (2/2)

- Die Portnummern sind in 3 Gruppen unterteilt:
    - 0 bis 1023 (Well Known Ports)
      - Diese sind Anwendungen fest zugeordnet und allgemein bekannt
    - 1024 bis 49151 (Registered Ports)
      - Anwendungsentwickler können sich Portnummern in diesem Bereich für eigene Anwendungen registrieren
    - 49152 bis 65535 (Private Ports)
      - Sind nicht registriert und können frei verwendet werden
- Verschiedene Anwendungen können im Betriebssystem gleichzeitig identische Portnummern verwenden, wenn Sie über unterschiedliche Transportprotokolle kommunizieren
  - Zudem gibt es Anwendungen, die Kommunikation via TCP und UDP über eine einzige Portnummer realisieren
  - Beispiel: Domain Name System – DNS (siehe Foliensatz 10)
- Die Well Known Ports und die Registered Ports werden durch die Internet Assigned Numbers Authority (IANA) vergeben
  - Unter Linux/UNIX existiert die Datei /etc/services
    - Hier sind Anwendungen (Dienste) den Portnummern zugeordnet
  - Unter Windows: %WINDIR%\system32\drivers\etc\services

# Sockets

- **Sockets** sind die plattformunabhängige, standardisierte **Schnittstelle** zwischen der Implementierung der Netzwerkprotokolle im Betriebssystem und den Anwendungen
- **Ein Socket besteht aus einer Portnummer und einer IP-Adresse**
- Man unterscheidet zwischen Stream Sockets und Datagram Sockets
  - **Stream Sockets** verwendeten das verbindungsorientierte TCP
  - **Datagram Sockets** verwendeten das verbindungslose UDP

Werkzeug(e) zur Kontrolle offener Ports und Sockets unter...

- Linux/UNIX: `netstat`, `lsof` und `nmap`
- Windows: `netstat`

Alternativen zu Sockets in der Interprozesskommunikation (IPC)

Pipes, Message Queues und gemeinsamer Speicher (Shared Memory)



# User Datagram Protocol (UDP)

- **Verbindungsloses Transportprotokoll**

- Datenübertragungen finden ohne vorherigen Verbindungsaufbau statt
- Einfacheres Protokoll als das verbindungsorientierte TCP
  - Nur für die Adressierung der Segmente zuständig
  - Es findet keine Sicherung der Datenübertragung statt
- Übertragungen werden nicht vom Empfänger beim Sender bestätigt
  - Segmente können bei der Übertragung verloren gehen
- Abhängig von der Anwendung, z.B. bei Videostreaming, ist das akzeptabel
  - Geht bei der Übertragung eines Videos via TCP ein Segment, also eine Bildinformation verloren, wird es neu angefordert
    - Es käme zu Aussetzern
  - Um das zu kompensieren, sind Wiedergabepuffer nötig
    - Speziell bei Videotelefonie versucht man aber die Puffer möglichst klein zu halten, weil diese zu Verzögerungen führen
  - Nutzt man UDP zur Übertragung eines Videos oder für Videotelefonie, geht beim Verlust eines Segments nur ein Bild verloren

# User Datagram Protocol (UDP)

- Maximale Größe eines UDP-Segments: 65.535 Bytes
  - Grund: Das **Länge**-Feld des UDP-Headers, das die Segmentlänge enthält, ist 16 Bits groß
    - Die maximal darstellbare Zahl mit 16 Bits ist 65.535
  - So große UDP-Segmente werden vom IP aber fragmentiert übertragen

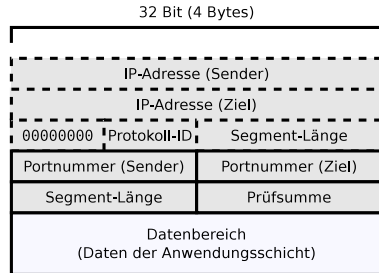


UDP-Standard: RFC 768 von 1980  
<http://tools.ietf.org/rfc/rfc768.txt>

Der Ablauf der Kommunikation via UDP und das Beispiel eines Servers und Clients mit Python wurde schon in Foliensatz 6 besprochen

# Aufbau von UDP-Segmenten

- Der UDP-Header besteht aus 4 je 16 Bit großen Datenfeldern
  - **Portnummer (Sender)**
    - Kann frei bleiben (Wert 0), wenn keine Antwort erforderlich ist
  - **Portnummer (Ziel)**
  - **Länge** des kompletten Segments (ohne Pseudo-Header)
  - **Prüfsumme** über das vollständige Segment (inklusive Pseudo-Header)
- Es wird ein Pseudo-Header erzeugt, der mit den IP-Adressen von Sender und Ziel auch Informationen der Vermittlungsschicht enthält
  - Protokoll-ID von UDP = 17
- Der Pseudo-Header wird nicht übertragen, geht aber in die Berechnung der Prüfsumme mit ein



Erinnern Sie sich an NAT aus Foliensatz 10...

Wird ein NAT-Gerät (Router) verwendet, muss dieses Gerät auch die Prüfsummen in UDP-Segmenten neu berechnen, es IP-Adressen übersetzt

# Transmission Control Protocol (TCP)

- **Verbindungsorientiertes Transportprotokoll**
- Erweitert das Vermittlungsprotokoll IP um die Zuverlässigkeit, die für viele Anwendungen gewünscht bzw. nötig ist
- Garantiert, dass Segmente **vollständig** und in der **korrekten Reihenfolge** ihr Ziel erreichen
  - Verlorene oder nicht bestätigte TCP-Segmente sendet der Sender erneut
- Eine TCP-Verbindung wird wie eine Datei geöffnet und geschlossen
  - Genau wie bei einer Datei wird die Position im Datenstrom exakt angegeben

TCP-Standard: RFC 793 von 1981  
<http://tools.ietf.org/rfc/rfc793.txt>

Der Ablauf der Kommunikation via TCP und das Beispiel eines Servers und Clients mit Python wurde schon in Foliensatz 6 besprochen

# Sequenznummern bei TCP

- TCP sieht Nutzdaten als unstrukturierten, aber geordneten Datenstrom
- **Sequenznummern** nummerieren den Strom der gesendeten Bytes
  - Die Sequenznummer eines Segments ist die Position des ersten Bytes des Segments im Bytestrom
- Beispiel
  - Der Sender unterteilt den Strom mit Anwendungsdaten in Segmente
    - Länge Datenstrom: 5.000 Bytes
    - MSS: 1.460 Bytes



## Einige Eckdaten...

**Maximum Transfer Unit (MTU):** Maximale Größe der IP-Pakete

MTU bei Ethernet = 1.500 Bytes, MTU bei PPPoE (z.B. DSL) = 1.492 Bytes

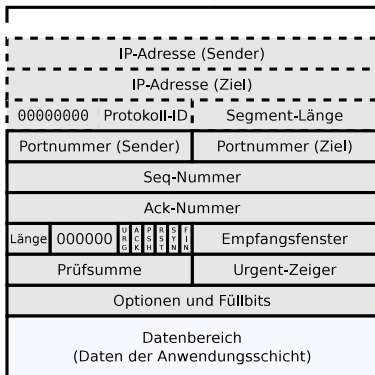
**Maximum Segment Size (MSS):** Maximale Segmentgröße

MSS = MTU - 40 Bytes für IPv4- und TCP-Header

# Aufbau von TCP-Segmenten (1/5)

- Ein TCP-Segment kann maximal 64 kB Nutzdaten (Daten der Anwendungsschicht) enthalten
  - Üblich sind kleinere Segmente ( $\leq 1.500$  Bytes bei Ethernet)

32 Bit (4 Bytes)



- Der Header von TCP-Segmenten ist komplexer im Vergleich zu UDP-Segmenten

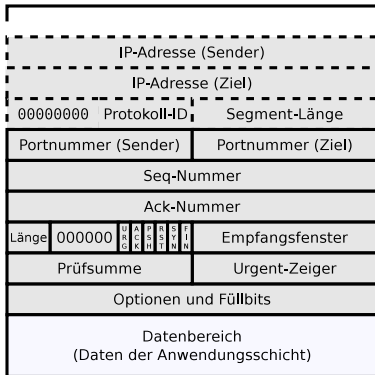
## Overhead

- Größe des TCP-Headers (ohne das Optionsfeld): nur 20 Bytes
- Größe des IP-Headers (ohne das Optionsfeld): auch nur 20 Bytes

⇒ Der Overhead, den die TCP- und IP-Header verursachen, ist bei einer IP-Paketgröße von mehreren kB gering

# Aufbau von TCP-Segmenten (2/5)

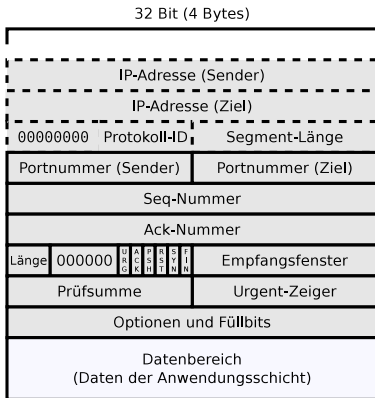
32 Bit (4 Bytes)



- Ein Datenfeld enthält die Portnummer des sendenden Prozesses
- Ein weiteres Datenfeld enthält die Portnummer des Prozesses, der das Segment empfangen soll
- **Seq-Nummer** enthält die Folgenummer (Sequenznummer) des aktuellen Segments
- **Ack-Nummer** enthält die Folgenummer des nächsten erwarteten Segments
- **Länge** enthält die Länge des TCP-Headers in 32-Bit-Worten, damit der Empfänger weiß, wo die Nutzdaten im TCP-Segment anfangen
  - Dieses Feld ist nötig, weil das Feld *Optionen und Füllbits* eine variable Länge (Vielfaches von 32 Bits) haben kann

# Aufbau von TCP-Segmenten (3/5)

- Das Datenfeld 000000 ist 6 Bits groß und wird nicht verwendet
  - Es hat den Wert Null



- Die 6 je 1 Bit großen Datenfelder sind für Verbindungsaufbau, Datenaustausch und Verbindungsabbau nötig
  - Im folgenden sind die Funktionen dieser Datenfelder jeweils so beschrieben, das sie den Wert 1 haben, also *gesetzt* sind

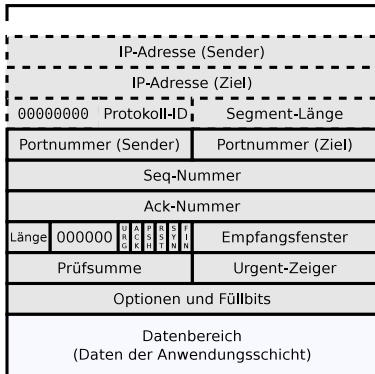
URG (Urgent) wird in der Vorlesung BSRN nicht behandelt

- **ACK** (Acknowledge)
  - Gibt an, dass die Bestätigungsnummer im Datenfeld **Ack-Nummer** gültig ist
  - Es wird also verwendet, um den Empfang von Segmenten zu bestätigen



# Aufbau von TCP-Segmenten (4/5)

32 Bit (4 Bytes)



PSH (Push) wird in der Vorlesung BSRN nicht behandelt

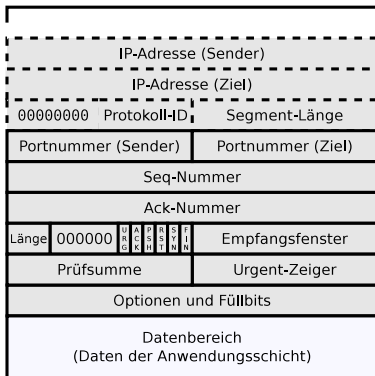
RST (Reset) wird in der Vorlesung BSRN nicht behandelt

- **SYN** (Synchronize)
  - Weist die Synchronisation der Sequenznummern an
  - Das initiiert den Verbindungsaufbau
- **FIN** (Finish)
  - Weist den Verbindungsabbau an und gibt an, dass der Sender keine Nutzdaten mehr schicken wird

Das **Empfangsfenster** wird in der Vorlesung BSRN nicht behandelt

# Aufbau von TCP-Segmenten (5/5)

32 Bit (4 Bytes)



- Genau wie bei UDP existiert auch für jedes TCP-Segment ein Pseudo-Header, der nicht übertragen wird
  - Dessen Datenfelder gehen aber inklusive regulärem TCP-Header und Nutzdaten in die Berechnung der **Prüfsumme** mit ein
  - Die **Protokoll-ID** von TCP ist die 6

Der **Urgent-Zeiger** wird in der Vorlesung BSRN nicht behandelt

Das Feld **Optionen und Füllbits** muss ein Vielfaches von 32 Bits groß sein und wird in dieser Vorlesung nicht behandelt

Erinnern Sie sich an NAT aus Foliensatz 8...

Wird ein NAT-Gerät (Router) verwendet, muss dieses Gerät auch die Prüfsummen in TCP-Segmenten neu berechnen, es IP-Adressen übersetzt

# Arbeitsweise von TCP

## Sie wissen bereits...

- Jedes Segment hat eine eindeutige Folgenummer (**Sequenznummer**)
- Die Sequenznummer eines Segments ist die Position des ersten Bytes des Segments im Bytestrom
- Anhand der Sequenznummer kann der Empfänger...
  - die Reihenfolge der Segmente korrigieren
  - doppelt angekommene Segmente aussortieren
- Die Länge eines Segments ist aus dem IP-Header bekannt
  - So werden Lücken im Datenstrom entdeckt und der Empfänger kann verlorene Segmente neu anfordern
- Beim Öffnen einer Verbindung (**Dreizege-Handshake**) tauschen beide Kommunikationspartner in drei Schritten Kontrollinformationen aus
  - So ist garantiert, dass der jeweilige Partner existiert und Daten annimmt

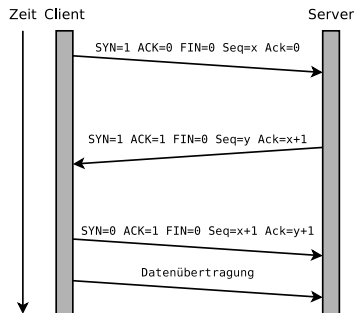
# TCP-Verbindungsaufbau (Dreiwege-Handshake)

- Der Server wartet passiv auf eine ankommende Verbindung

- 1 Client sendet ein Segment mit  $\text{SYN}=1$  und fordert damit zur Synchronisation der Folgenummern auf  
 $\Rightarrow$  *Synchronize*

- 2 Server sendet als Bestätigung ein Segment mit  $\text{ACK}=1$  und fordert mit  $\text{SYN}=1$  seinerseits zur Synchronisation der Folgenummern auf  
 $\Rightarrow$  *Synchronize Acknowledge*

- 3 Client bestätigt mit einem Segment mit  $\text{ACK}=1$  und die Verbindung steht  
 $\Rightarrow$  *Acknowledge*



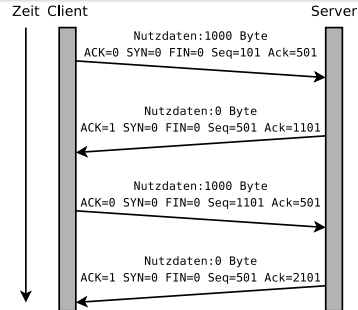
- Die Anfangs-Sequenznummern ( $x$  und  $y$ ) werden zufällig bestimmt
- Beim Verbindungsaufbau werden keine Nutzdaten ausgetauscht!

# TCP-Datenübertragung

Um eine Datenübertragung zu zeigen, sind für die **Seq-Nummer** (Folgenummer aktuelles Segment) und die **Ack-Nummer** (Folgenummer nächstes erwartetes Segment) konkrete Werte nötig

- Im Beispiel ist zu Beginn des Dreiwege-Handshake die Folgenummer des Clients  $x=100$  und die des Servers  $y=500$
- Nach Abschluss des Dreiwege-Handshake:  $x=101$  und  $y=501$

- Client überträgt 1000 Byte Nutzdaten
- Server bestätigt mit  $ACK=1$  die empfangenen Nutzdaten und fordert mit der Ack-Nummer das nächste Segment an
- Client überträgt weitere 1000 Byte Nutzdaten
- Server bestätigt mit  $ACK=1$  die empfangenen Nutzdaten und fordert mit der Ack-Nummer das nächste Segment an

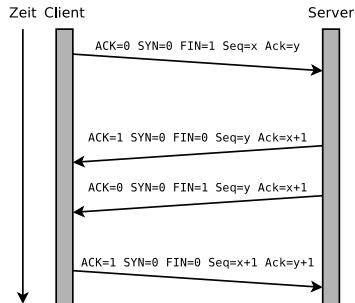


Im Beispiel zählt der Server seine Seq-Nummer nicht hoch, weil er keine Nutzdaten an den Client überträgt

# TCP-Verbindungsabbau

- Der Verbindungsabbau ist dem Verbindungsaufbau ähnlich
- Statt des SYN-Bit kommt das FIN-Bit zum Einsatz, das anzeigt, dass keine Nutzdaten mehr vom Sender kommen

- 1 Client sendet den Abbauwunsch mit FIN=1
- 2 Server sendet eine Bestätigung mit ACK=1
- 3 Server sendet den Abbauwunsch mit FIN=1
- 4 Client sendet eine Bestätigung mit ACK=1



- Beim Verbindungsabbau werden keine Nutzdaten ausgetauscht

# Denial of Service-Attacken via SYN-Flood

- Ziel des Angriffs: Dienste oder Server unerreichbar machen
- Ein Client sendet viele Verbindungsanfragen (**SYN**), antwortet aber nicht auf die Bestätigungen (**SYN ACK**) des Servers mit **ACK**
- Der Server wartet einige Zeit auf die Bestätigung des Clients
  - Es könnten ja Netzwerkprobleme die Bestätigung verzögern
  - Während dieser Zeit werden die Client-Adresse und der Status der unvollständigen Verbindung im Speicher des Netzwerkstacks gehalten
- Durch das Fluten des Servers mit Verbindungsanfragen wird die Tabelle mit den TCP-Verbindungen im Netzwerkstack komplett gefüllt  
⇒ Der Server kann keine neuen Verbindungen mehr aufbauen
- Der Speicherverbrauch auf dem Server kann so groß werden, dass der Hauptspeicher komplett gefüllt wird und der Server abstürzt
- Gegenmaßnahme: Echtzeitanalyse des Netzwerks durch intelligente Firewalls