

# Netzwerke und Kommunikation

## B-LS-MI 004

### Layer-4 UDP & TCP

rolf.schmutz@fhnw.ch

FHNW

14. Oktober 2020

2020/NDK05/nd06.tex

# Ziele

- Sie kennen die Transportschichtprotokolle UDP und TCP und geeignete Anwendungen
- Sie kennen die Software-Abstraktion “Socket” und das dazugehörige demultiplexing auf dem System
- Sie können Verbindungen auf dem System identifizieren

# Aufgaben der Schichten

Adressierung/Demultiplexing:

- Layer-4: Prozess-zu-Prozess<sup>1</sup>
- Layer-3: Host-zu-Host<sup>2</sup>
- Layer-2: Host-zu-*lokalem-Host/Router*

---

<sup>1</sup>Programm-zu-Programm, z.B. Webbrowser-zu-Webserver

<sup>2</sup>end-to-end

## Layer-4: Transportschicht (1/3)

- Die Schicht 4 führt eine Abstraktion für Kommunikationskanäle ein, die die unterliegende Paketschicht verbirgt
- 1 Es gibt einen verbindungslosen “Telegrammdienst” (UDP) für kurze und/oder “einweg” Meldungen<sup>3</sup>
  - 2 ... und einen verbindungsorientierten, bidirektionalen Dienst mit garantierter Sequenz<sup>4</sup>

### Abstraktion

**beides sind “Illusionen”, die die paketorientierte Arbeitsweise von IP verbergen**

auf beiden Endgeräten muss ein Verbindungsstatus gepflegt werden

---

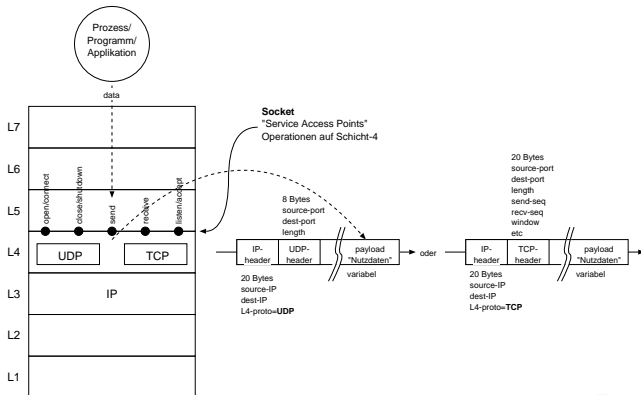
<sup>3</sup> . . . ziemlich genaue Analogie

<sup>4</sup> analog z.B. einer Telefonverbindung

# Layer-4: Transportschicht Services (2/3)

Einige SAP/Services:

- Server: SAP *passive-open* accept/recv (warte auf Anfragen)
- Client: SAP *active-open* connect/sendto (startet eine Anfrage)
- Beide: SAPI *send, receive, close* (Datenkommunikation)



## Layer-4: Transportschicht Abstraktion “Socket” (3/3)

- Software-Abstraktion eines Kommunikationsendpunkts “Berkeley Socket”<sup>5</sup>
- der Kommunikationskanal kann wie eine Datei angesprochen werden<sup>6</sup>

```
import socket
```

```
s = socket.socket() # "s" may subsequently be used like a file-object for reading, writing
s.connect(('www.google.com', 80)) # use defaults AF_INET, SOCK_STREAM=TCP
```

```
s.send(b"GET HTTP/1.0\n\n")
```

```
r = s.recv(10000) # buffer-size
```

```
print(str(r))
```

```
s.close()
```

<sup>5</sup>von UC Berkley, BSD “Berkeley Software Distribution” UNIX

<sup>6</sup>*read=recv* und *write=send*. Bei TCP zusätzlich *open=connect* und *close*



# UDP: User Datagram Protocol

- kann für kurze Einwegmeldungen<sup>7</sup> wie z.B. Systemlog<sup>8</sup>
- oder auch für bidirektionale Konversation<sup>9</sup> wie z.B. DNS/Verzeichnisdienst<sup>10</sup> verwendet werden
- es ist Aufgabe der Applikation<sup>11</sup> Antwort-Datagramme zu senden – UDP selbst “kennt” das jeweilige Schicht-7 Protokoll nicht
- unterstützt *Multicasting* – senden von Daten an viele Hosts gleichzeitig
- die Bezeichnung für eine Dateneinheit (Telegramm) ist *datagram*

## Telegrammdienst

**Die jeweiligen Applikationen/Programme<sup>a</sup> müssen die Quittierung, Wiederholung, Zuweisung von Meldungen selber sicherstellen**

<sup>a</sup> client und server

<sup>7</sup> d.h. ohne Bestätigungsmeldung, best-effort

<sup>8</sup> Windows: *Eventlog*, Transkript

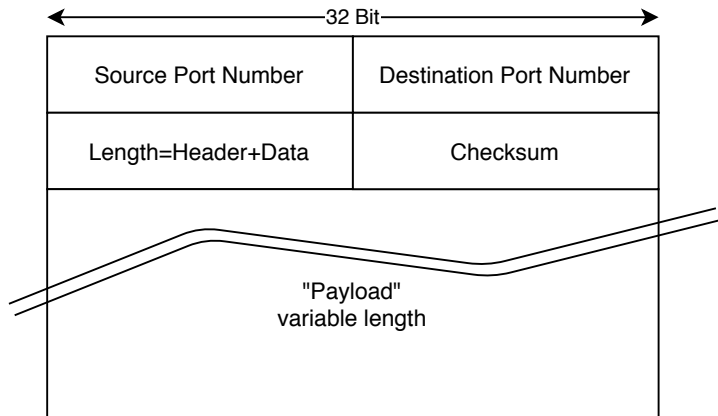
<sup>9</sup> request und reply

<sup>10</sup> ...damit Sie [www.eff.org](http://www.eff.org) eingeben können und DNS findet dann die IP-Adresse 64.147.188.3 dazu

<sup>11</sup> Prozess/“Programm”, auf Server- und Client-Seite

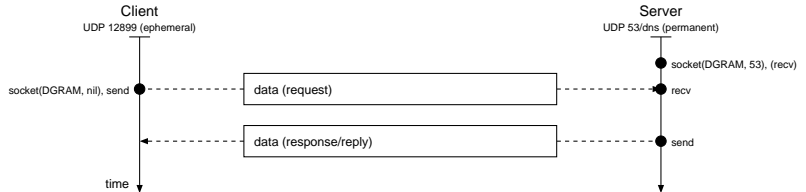


# UDP Header

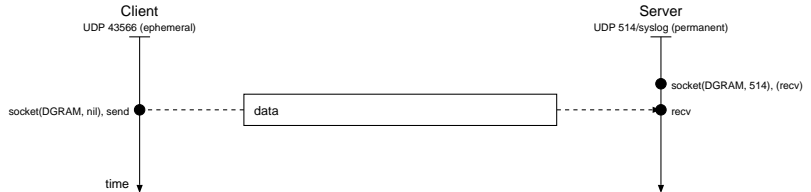


- 16 Bit Port Nummern erlauben  $2^{16} = 65536$  mögliche Endpunkte
- keine Status-Flags, etc
- Header ist sehr "kompakt", nur 16 octets/Bytes "overhead"

# UDP Communication



OR





# TCP: Transmission Control Protocol 1/5

- Zweiweg<sup>12</sup> verbindungsorientierte Kommunikation
- garantierte Sequenz der Daten<sup>13</sup>
- verlorene Pakete werden neu gesendet
- *Flusskontrolle* – Empfänger kann “stop” oder “langsamer senden” verlangen
- die Bezeichnung für eine Dateneinheit ist *segment* – allerdings ist die Abstraktion für die Software ein *stream* (Datenstrom)

## Verbindungsorientierter Dienst

### Transparente<sup>a</sup> bidirektionale (Richtungsgetrennt) Verbindung<sup>b</sup>

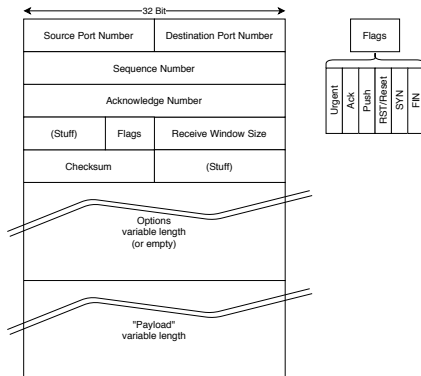
<sup>a</sup>d.h. die Client- und Server-Applikationen kümmern sich nicht um Paketwiederholungen, Sequenz, etc

<sup>b</sup>das ist eine nur eine “Illusion” – die darunterliegende Schicht IP ist nicht Verbindungsorientiert

<sup>12</sup>bidirektional

<sup>13</sup>auch wenn sich Pakete im Internet “überholen”

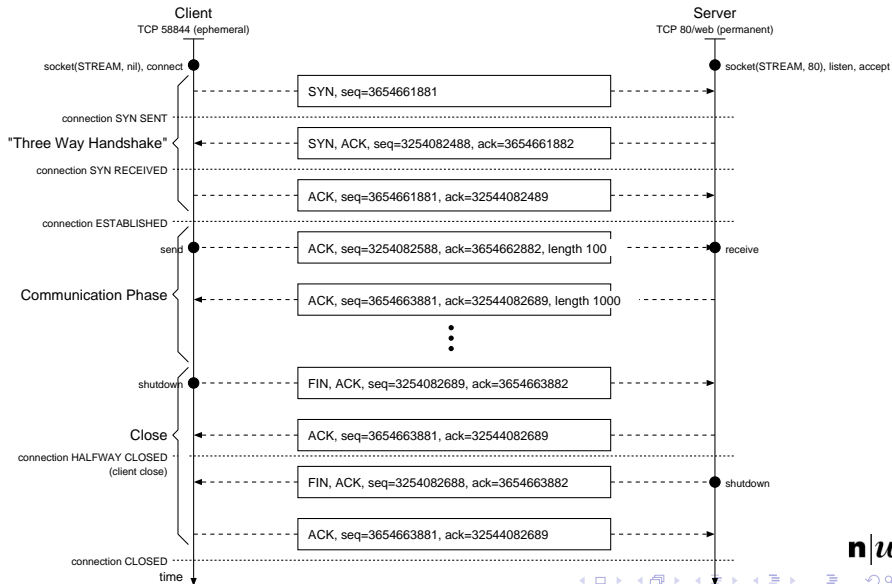
# TCP Header



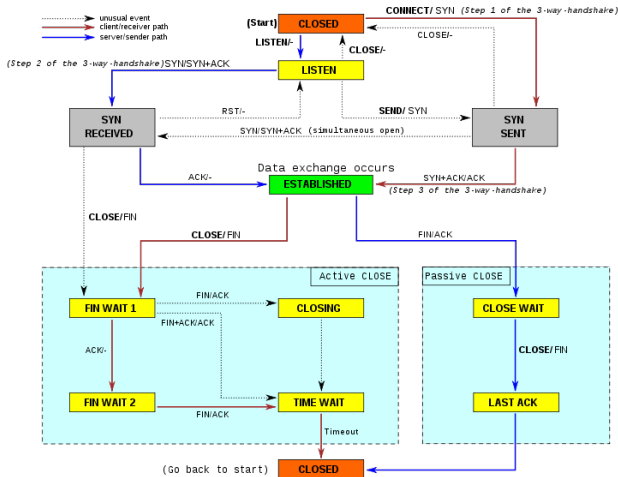
- 16 Bit Port Nummern erlauben  $2^{16} = 65536$  mögliche Endpunkte
- Status-Flags: ermöglichen Abstraktion in einer Status-Maschine
- Sequenz-Nummer erlaubt die Positionierung innerhalb des Datenstroms<sup>14</sup>, Acknowledge/Bestätigung ist eine Antwort auf die Sequenz-Nummer des Kommunikationspartners
- Window-Size optimiert den Datentransfer mit verzögerter Quittung/Acknowledge
- Header ist *mindestens* 10 octets/Bytes

<sup>14</sup> für "reassembly" auf Empfängerseite

# TCP Handshake, Session 2/5



# TCP Status-Diagramm<sup>15</sup> 4/5



<sup>15</sup> [http://upload.wikimedia.org/wikipedia/commons/thumb/a/a2/Tcp\\_state\\_diagram\\_fixed.svg/796px-Tcp\\_state\\_diagram\\_fixed.svg.png](http://upload.wikimedia.org/wikipedia/commons/thumb/a/a2/Tcp_state_diagram_fixed.svg/796px-Tcp_state_diagram_fixed.svg.png)

# Socket-Status in Realität

- SYN-SENT und SYN-RECEIVED sieht man in der Realität selten (kurzfristig)  
– ausser in der Client-Rolle bei “geblockter” Verbindung (Firewall)
- die WAIT-Varianten sieht man öfter – eine Indikation der Connection-Rate
- bei TCP natürlich ESTABLISHED, bei UDP natürlich *nicht*

```

root@zaphod:~# netstat -tunap4
Active Internet connections (servers and established)

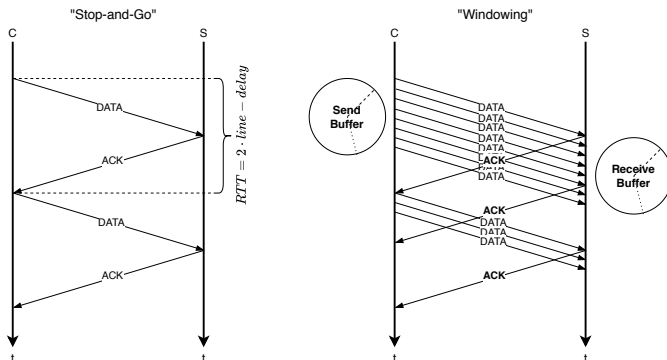
```

Proto	Recv-Q	Send-Q	Local Address	Foreign Address	State	PID/Program name
tcp	0	0	0.0.0.0:993	0.0.0.0:*	LISTEN	7720/imap-login
tcp	0	0	0.0.0.0:80	0.0.0.0:*	LISTEN	19994/lighttpd
tcp	0	0	127.0.0.1:53	0.0.0.0:*	LISTEN	32004/named
tcp	0	0	0.0.0.0:22	0.0.0.0:*	LISTEN	3097/sshd
tcp	0	0	0.0.0.0:25	0.0.0.0:*	LISTEN	1602/master
tcp	0	0	0.0.0.0:443	0.0.0.0:*	LISTEN	19994/lighttpd
tcp	0	0	188.40.65.199:22	77.56.89.75:45753	ESTABLISHED	18655/sshd: tunnel
tcp	0	0	188.40.65.199:22	212.60.51.243:40469	ESTABLISHED	5604/sshd: tunnel
tcp	0	0	188.40.65.199:22	212.60.51.243:46973	ESTABLISHED	24007/sshd: tunnel
tcp	0	3248	188.40.65.199:22	77.56.89.75:52550	ESTABLISHED	24992/sshd: rschmutz
tcp	1	0	188.40.65.199:80	77.56.89.75:51856	CLOSE_WAIT	19994/lighttpd
udp	0	0	188.40.65.199:53	0.0.0.0:*		32004/named
udp	0	0	188.40.65.199:123	0.0.0.0:*		3057/ntp



# TCP "Windowing"

TCP implementiert ARQ<sup>16</sup> mittels eines effizienten Window-Verfahrens



- Dies bedingt jedoch einen Sende- und Empfangs-Buffer/Zwischenspeicher
- mit dem "Windowsize" gibt der Empfänger dem Sender die Menge an "unacked"-Daten bekannt, die empfangen werden können

<sup>16</sup> "automatic repeat request"

# TCP tcpdump<sup>17</sup> (edited) 5/5

--- three-way handshake ---


```
19:09:56.361262 IP 10.202.5.121.63505 > 188.40.65.199.80: Flags [S], seq 1704735491, win 65535, length 0
19:09:56.384815 IP 188.40.65.199.80 > 10.202.5.121.63505: Flags [S.], seq 4146040110, ack 1704735492,
    win 5792, length 0
19:09:56.384871 IP 10.202.5.121.63505 > 188.40.65.199.80: Flags [.], ack 4146040111, win 33304, length 0
```

--- communication phase ---

```
19:10:00.891376 IP 10.202.5.121.63505 > 188.40.65.199.80: Flags [P.], seq 1704735492:1704735509,
    ack 4146040111, win 33304, length 17
19:10:00.915173 IP 188.40.65.199.80 > 10.202.5.121.63505: Flags [.], ack 1704735509, win 46, length 0
19:10:06.987161 IP 10.202.5.121.63505 > 188.40.65.199.80: Flags [P.], seq 1704735509:1704735533,
    ack 4146040111, win 33304, length 24
19:10:07.010497 IP 188.40.65.199.80 > 10.202.5.121.63505: Flags [.], ack 1704735533, win 46, length 0
19:10:07.531102 IP 10.202.5.121.63505 > 188.40.65.199.80: Flags [P.], seq 1704735533:1704735535,
    ack 4146040111, win 33304, length 2
19:10:07.555122 IP 188.40.65.199.80 > 10.202.5.121.63505: Flags [.], ack 1704735535, win 46, length 0
19:10:07.555127 IP 188.40.65.199.80 > 10.202.5.121.63505: Flags [P.], seq 4146040348,
    ack 1704735535, win 46, length 237
19:10:07.555182 IP 10.202.5.121.63505 > 188.40.65.199.80: Flags [.], ack 4146040348, win 33185, length 0
```

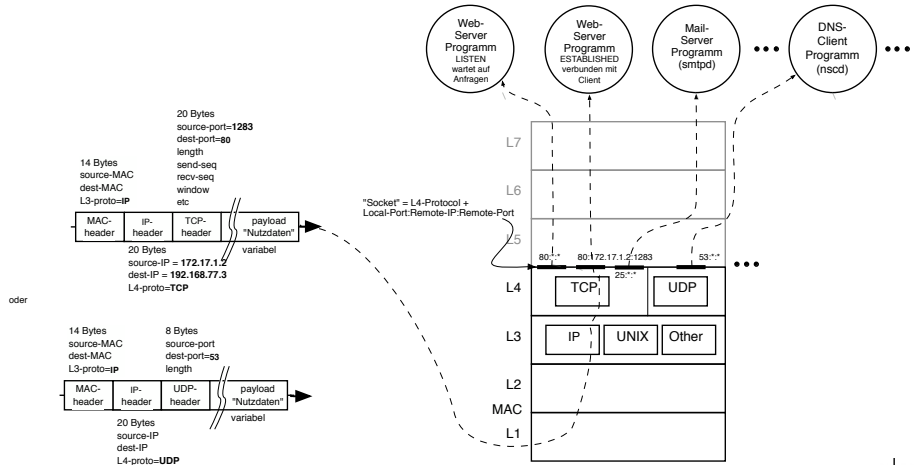
--- shutdown ---

```
19:10:12.792188 IP 188.40.65.199.80 > 10.202.5.121.63505: Flags [F.], seq 4146040348,
    ack 1704735535, win 46, length 0
19:10:12.792244 IP 10.202.5.121.63505 > 188.40.65.199.80: Flags [.], ack 4146040349, win 33304, length 0
19:10:12.792341 IP 10.202.5.121.63505 > 188.40.65.199.80: Flags [F.], seq 1704735535,
    ack 4146040349, win 33304, length 0
19:10:12.815841 IP 188.40.65.199.80 > 10.202.5.121.63505: Flags [.], ack 1704735536, win 46, length 0
```

<sup>17</sup> `sudo tcpdump -v -nnn -S -i en1 tcp port 80 and ip host zaphod und dann nc zaphod 80` 

# Layer-4: Demultiplexing

für das Demultiplexing/Zuweisung an Programm/Socket wird bei TCP das 5-Tuple {protocol, local-ip, local-port, remote-ip, remote-port}, bei UDP nur {protocol, local-ip, local-port} verwendet<sup>18</sup>



<sup>18</sup> bei UDP muss das Server-Programm die einzelnen Meldungen nach Absender demultiplexen/zuzuweisen

# Port Nummern<sup>23</sup> $\approx$ Dienst

- um einen bestimmten Dienst<sup>19</sup> anzusprechen müssen die die entsprechenden Portnummern bekannt sein
- Systemseitig werden anstatt Portnummern oft symbolische Namen benutzt<sup>20</sup>, Windows: C:
- Portnummern werden von IANA<sup>21</sup> verwaltet  
es gibt die

1 *well-known-services*<sup>22</sup> 0 bis 1023

2 *registered ports* 1024 bis 49151: darin finden sich bekannte Dienste (“Server-side”, *permanent*) aber auch “Client-side” (*ephemeral* Ports)

---

<sup>19</sup> z.B. Web oder Mail

<sup>20</sup> UNIX: `/etc/services` und `getent services mail` oder `getent services 25`

<sup>21</sup> Internet Assigned Numbers Authority, <http://www.iana.org/assignments/port-numbers>

<sup>22</sup> “WKS” auch bekannt als “low-ports”

<sup>23</sup> [http://en.wikipedia.org/wiki/TCP\\_and\\_UDP\\_port\\_numbers](http://en.wikipedia.org/wiki/TCP_and_UDP_port_numbers)

# Command Line Tools

- Socket Status, “offene Ports”: `netstat -an` (alle sockets)
- TCP Verbindungstest: `telnet host port`

# References

- Internet Standards: <http://tools.ietf.org/html/rfc1280>
- UDP:  
RFC <http://tools.ietf.org/html/rfc768> und  
Standard <http://tools.ietf.org/html/std6>
- TCP:  
RFC <http://tools.ietf.org/html/rfc793> und  
Standard <http://tools.ietf.org/html/std7>
- Socket: [http://en.wikipedia.org/wiki/Internet\\_socket](http://en.wikipedia.org/wiki/Internet_socket)
- Port Nummern: [http://en.wikipedia.org/wiki/TCP\\_and\\_UDP\\_port\\_numbers](http://en.wikipedia.org/wiki/TCP_and_UDP_port_numbers) und  
<http://www.iana.org/assignments/port-numbers>