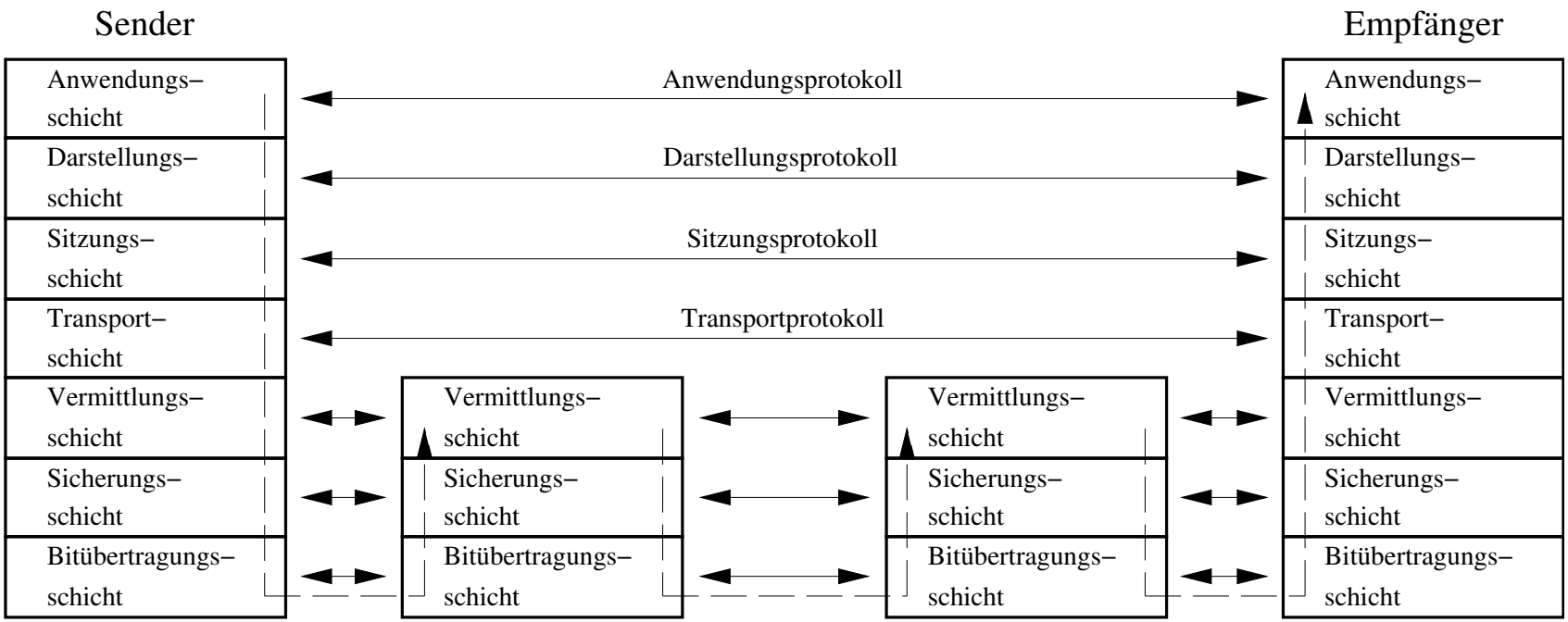
Das Transportprotokoll ist ein Ende-zu-Ende-Protokoll zwischen Quell- und Zielrechner.



Internet Protocol Stack

Schicht 5-7	ssh	SMTP	Telnet	FTP	HTTP	NFS/ DNS,	SNMP	NTP
4	TCP						UDP	
3	IP							
1-2	Schicht 1 + 2							

Beispiele: TCP (Transmission Control Protocol),
UDP (User Datagram Protocol)

Wesentliche Aufgabe der Transportschicht ist die Erhöhung der Dienstqualität des Vermittlungsdienstes:

- ggf. Paketverlust überwachen und korrigieren,
- ggf. Auf- und Abbau von Tranportverbindungen
- Flußkontrolle zwischen Quell- und Zielrechner
- ggf. Aufteilen der Nachricht in kleinere Dateneinheiten für Schicht 3
- Multiplexing oder Splitting auf Verbindungen der Schicht 3

10.1 UDP (USER Datagram Protocol)

- Jon Postel, RFC 768, 1980
- Benutzerschnittstelle zum IP
- verbindungsloses Transportprotokoll (Schicht 4),
- keine Fehlerkontrolle $\Longrightarrow$ Pakete können verloren gehen, dupliziert werden
- Postfach-Adressierung an sogenannte Ports

Woher kennt der Client den Server-Port?

1. Well-Known Port Assignment für Standardanwendungen.

In /etc/services sind die Well Known Port Numbers aufgelistet, z.B.:

```
# WELL KNOWN PORT NUMBERS
#
ftp-data          20/tcp    #File Transfer [Default Data]
ftp               21/tcp    #File Transfer [Control]
telnet            23/tcp
smtp              25/tcp    mail #Simple Mail Transfer
http              80/tcp    www www-http #World Wide Web HTTP
snmp             161/udp    #Simple Network Management Protocol
```

2. Dynamic Binding: Beim Erzeugen des Sockets wird dem Socket dynamisch vom Betriebssystem der nächste freie Port zugewiesen.

1. Sinnvoll auf Client-Seite!
2. Aber: **Woher kennt der Client den Server-Port?**

   $\Longrightarrow$ Auskunftsdienst

   Der Server meldet sich mit seiner Portnummer bei einem Auskunftsdienst an.
   Der Auskunftsdienst wiederum muß eine Well-Known Port Number besitzen.

Beispiel: Sun *RPC* benutzt portmapper: `programm name` $\mapsto$ `port`
Bei Java RMI heißt der Auskunftsdienst RMIRegistry.

Socket-Adressen

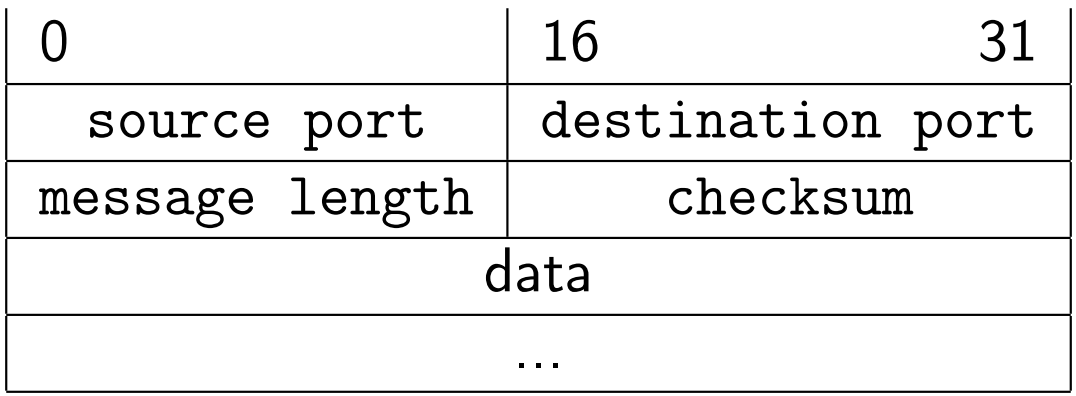
- Adressen werden in einer allgemeinen Struktur angegeben, die aus einem Feld für die Protokollfamilie, einer Portnummer und einer für diese Protokollfamilie spezifischen Adresse besteht:

```
struct sockaddr_in {
    sa_family_t    sin_family; /* address family: AF_INET */
    u_int16_t      sin_port;   /* port in network byte order */
    struct in_addr  sin_addr;  /* AF_INET: internet address */
};
```

Wie erfährt der Server den Port des Clients?

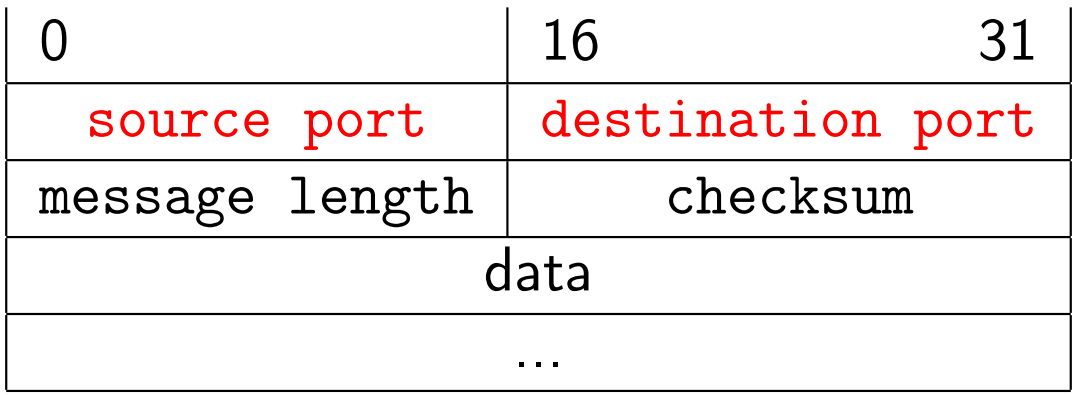
Pakete heißen User Datagram.

Paketformat:



Pakete heißen User Datagram.

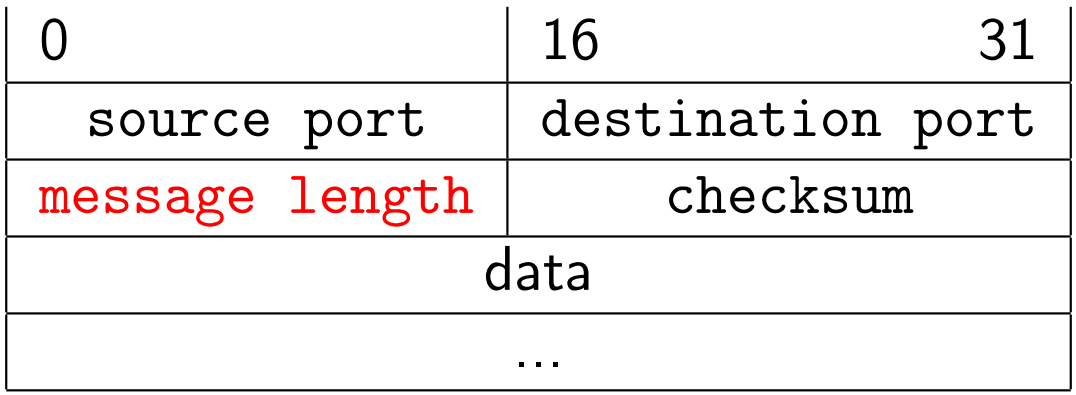
Paketformat:



`source port`: Verwendung ist optional, falls Antworten geschickt werden sollen, sonst `source port` $= 0 \ldots 0$

Pakete heißen User Datagram.

Paketformat:

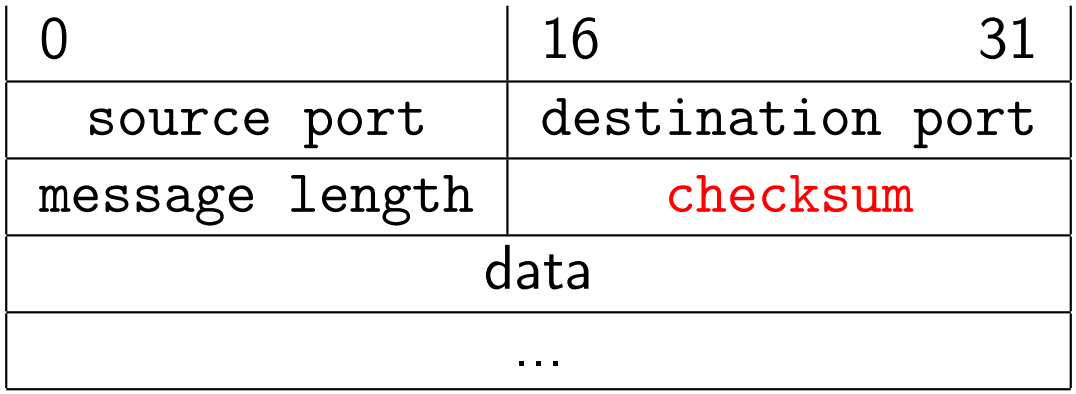


Paketlänge (Header + Daten):
8 Byte $\leq$ `message length` $\leq 2^{16} - 1 = 65\,535$ Byte

$\Longrightarrow$ maximale UDP Datagramgröße!

Pakete heißen User Datagram.

Paketformat:



Verwendung der Prüfsumme (`checksum`) ist optional:
`checksum` $= 0 \ldots 0 \;\hat{=}$ Prüfsumme wird nicht benutzt

Prüfsumme gemäß Einerkomplement wird berechnet für:

UDP-Header mit `checksum` = 0	Pseudoheader	`user data`	Padding

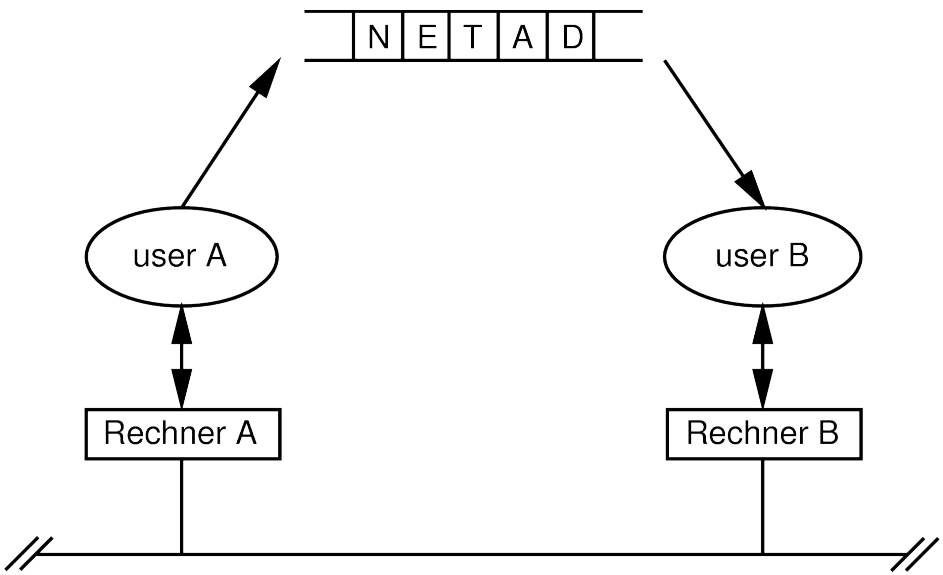
Padding: auf Vielfaches von 16 Bit mit Nullen auffüllen

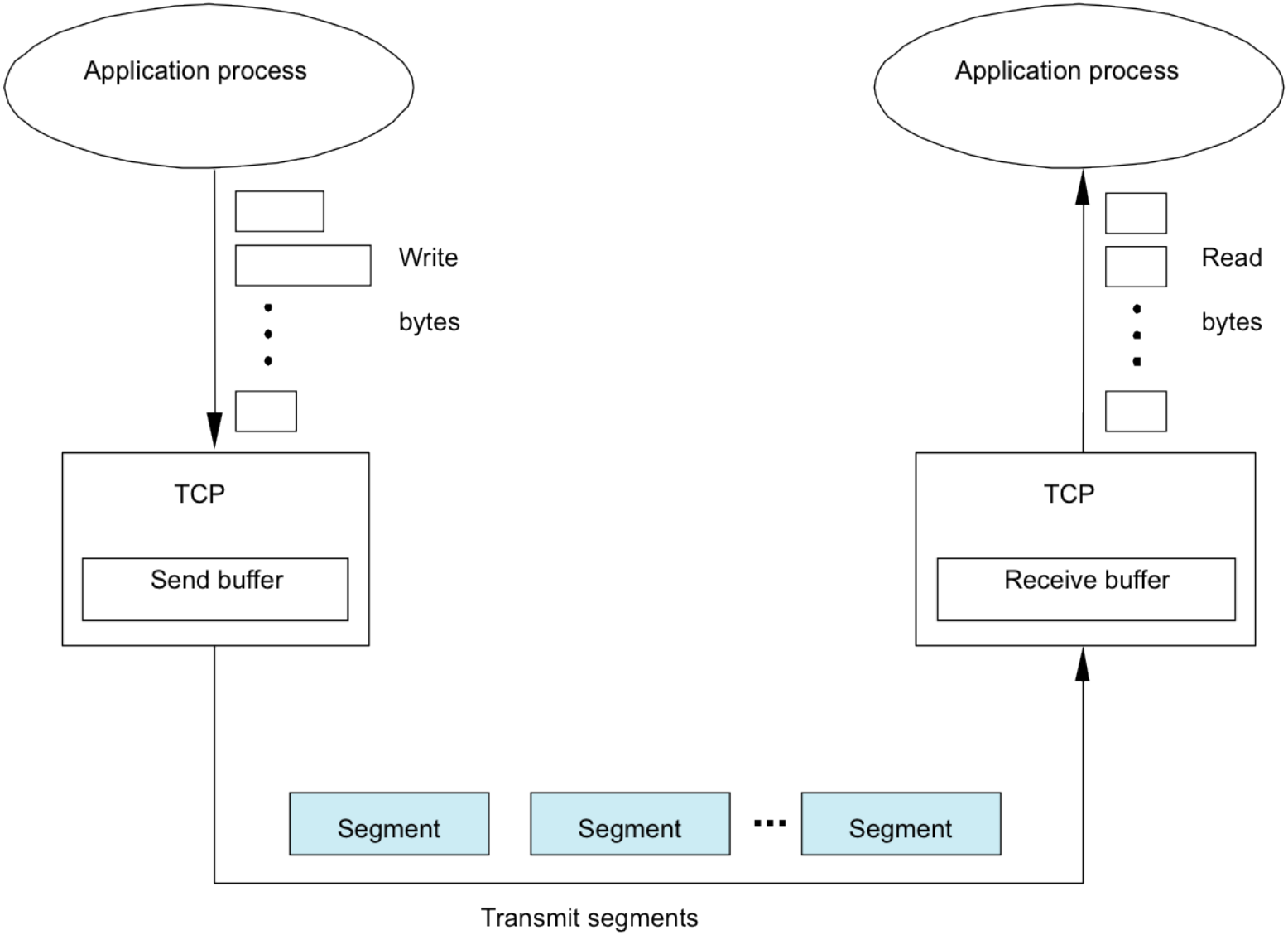
UDP–Pseudoheader:

0	8	16 31
SOURCE IP ADDRESS		
DESTINATION IP ADDRESS		
00000000	PROTOCOL NUMBER	UDP LENGTH

10.2 TCP (Transmission Control Protocol)

- RFC 793 (Jon Postel, 1981), 1122, 1323, ...,
  Neuauflage der Spezifikation in RFC 9293, August 2022
- **zuverlässiges**, verbindungsorientiertes Transportprotokoll,
- logische Verbindung zwischen Sender und Empfänger,
- bestätigter Verbindungsauf-/abbau
- zuverlässiges, geordnetes Übertragen von Daten mittels *PAR*-Protokoll



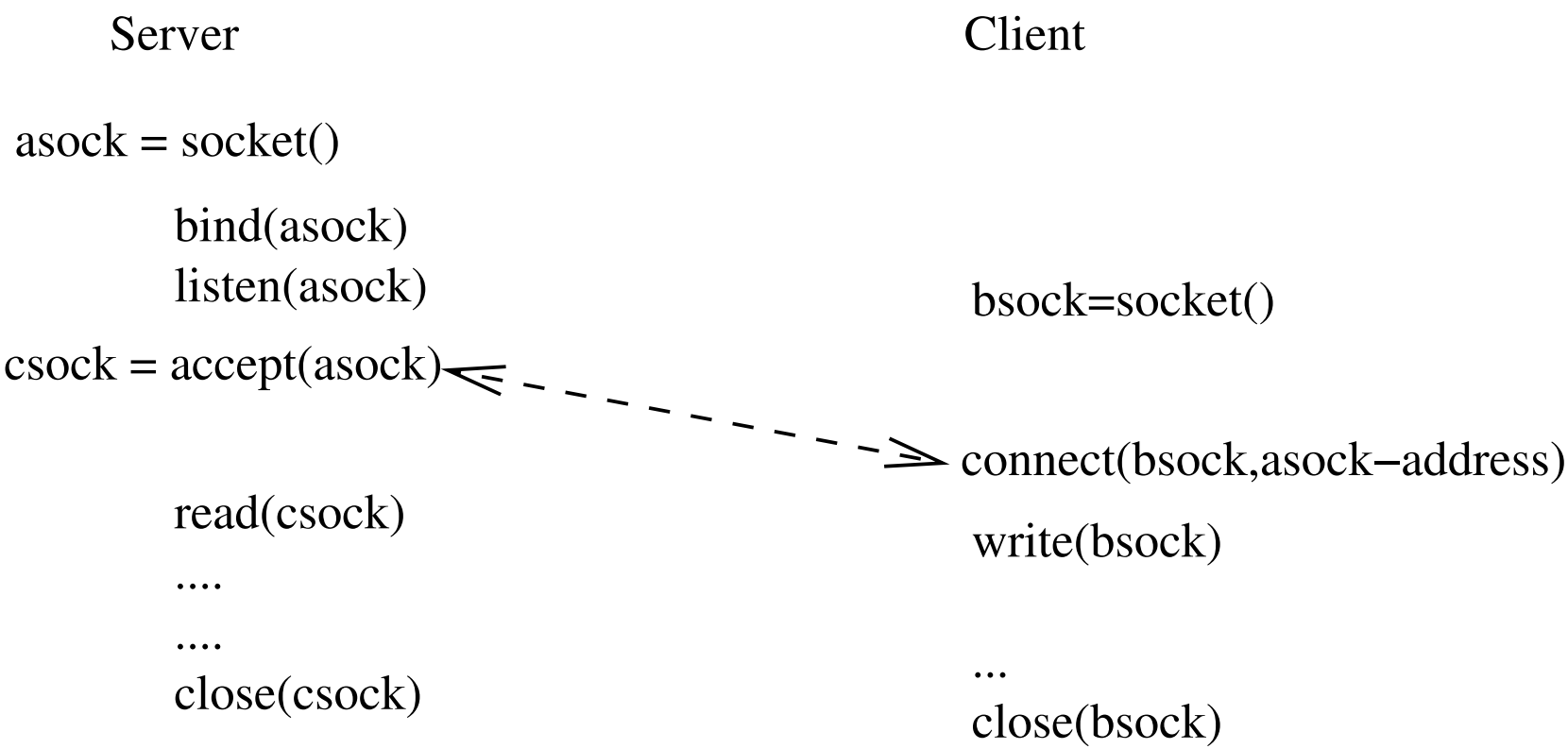

Application process
Application process
Write
bytes
Read
bytes
TCP
Send buffer
TCP
Receive buffer
Segment
Segment
Segment
Transmit segments

- Fehlerkontrolle wird nicht für Pakete, sondern für zu übertragenden Bytestrom durchgeführt: Bytes werden durchnummeriert.
  (Sliding Window Protocol mittels Quittungen und Timeout-Kontrolle)
- Ende-zu-Ende-Flußkontrolle über variable Empfangsfenstergröße
- “out-of-band”-Anzeige von dringenden Daten

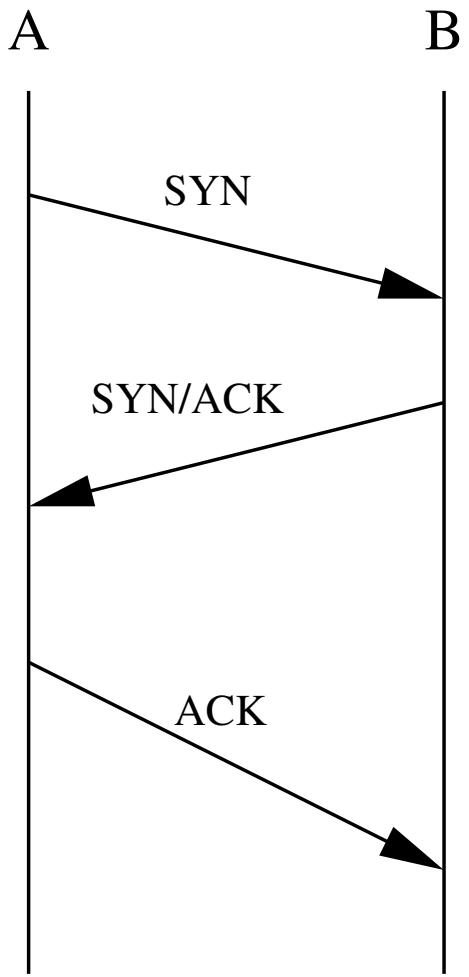
– nur Punkt-zu-Punkt-Kommunikation (keine Multicasts!!!)

Wiederholung: Socket-Aufrufe für verbindungsorientiertes Protokoll

- Beim verbindungsorientierten Protokoll wird zunächst von einer Seite ein Socket geöffnet, über den Verbindungswünsche entgegen genommen werden.
- Der `accept()`-Aufruf blockiert den Prozeß, bis eine Verbindung etabliert ist und liefert einen neuen Socket für diese Verbindung.
- Die `read()` und `write()` Aufrufe sind blockierend.
- Nach der Auflösung der Verbindung kann mit einem erneuten Aufruf von `accept()` eine weitere Verbindung entgegen genommen werden.


Server
Client
asock = socket()
bind(asock)
listen(asock)
bsock=socket()
csock = accept(asock)
connect(bsock,asock–address)
read(csock)
write(bsock)
....
....
...
close(csock)
close(bsock)

A initiiert die Verbindung (Client):



3-way Handshake

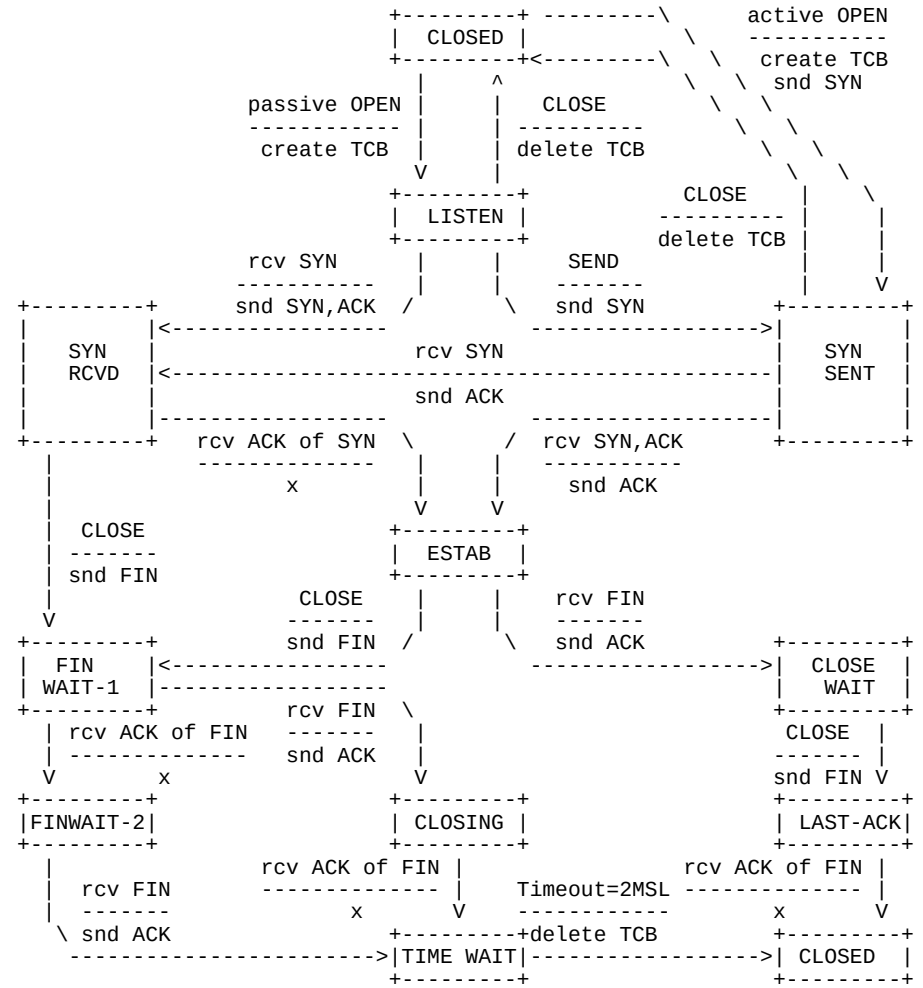
Ablauf:

1. SYN Paket: Connection Setup
   Nachricht mit gesetztem SYN-Bit (Synchronize) und gesetzter Sequenznummer. Die Sequenznummer ist eine frei gewählte Anfangsnummer des Bytestroms (zufällig gewählt): SEQ = x
2. SYN/ACK Paket: Connection accepted, d.h. SYN-Bit und ACK-Bit gesetzt
   - Die Sequenznummer wird bestätigt: ACK = x+1
   - und ebenfalls eine Sequenznummer für den Rückkanal gewählt: SEQ = y
3. ACK Paket: Connection Confirmation mit gesetztem ACK-Bit ACK = y+1, ggf. auch mit Daten (Payload)

Weiterhin wird ausgetauscht:

- Receiver Maximum Segment Size
- Initial Window

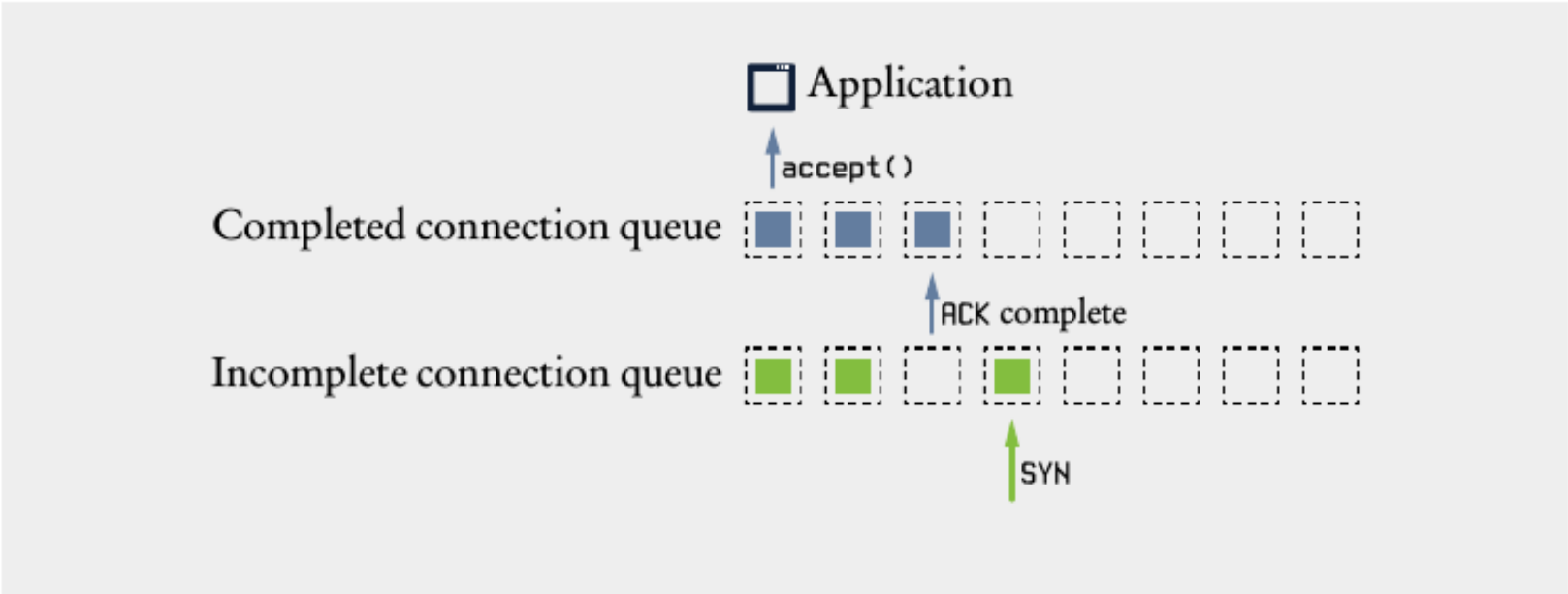
aus dem RFC 793



TCP Connection State Diagram
Figure 6.

Backlog-Queue

Figure 6.1 Example backlog queues



Quelle: Jörg Jung, Dissertation, Seite 184, UP 2015.

Die Backlog-Queue teilt sich in Completed (*Ack Queue*) und Incomplete Connection Queue auf. Trifft ein SYN-Paket ein, wird es in der Incomplete Connection Queue zwischengespeichert. Trifft das Ack-Paket ein, wird es in die Completed Queue eingetragen. Die Summe beider Warteschlangenlängen darf das Limit für die Backlog-Queue nicht übersteigen.

Linux Backlog Implementierung:

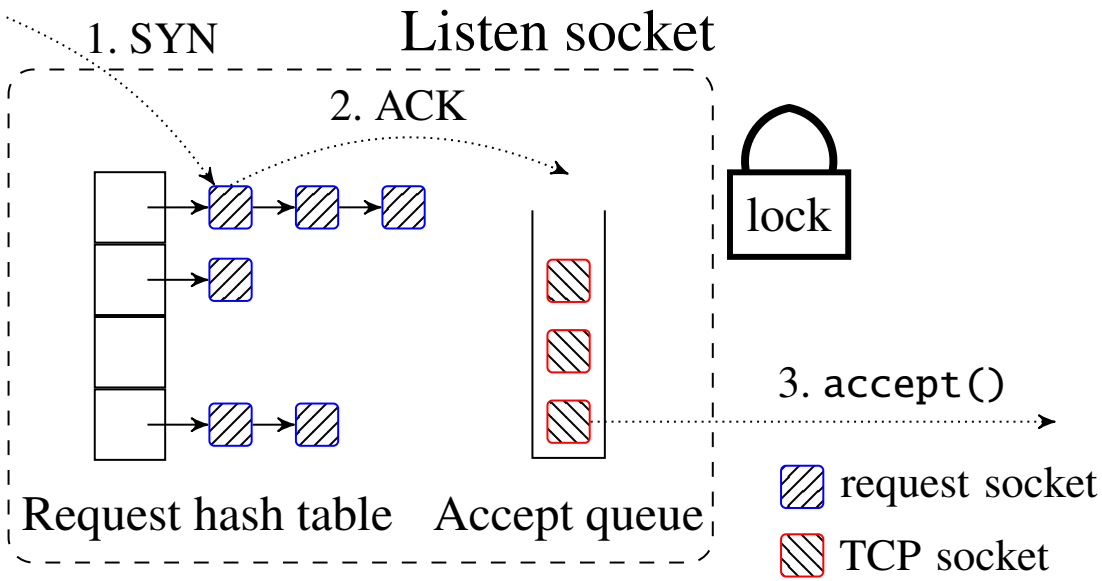
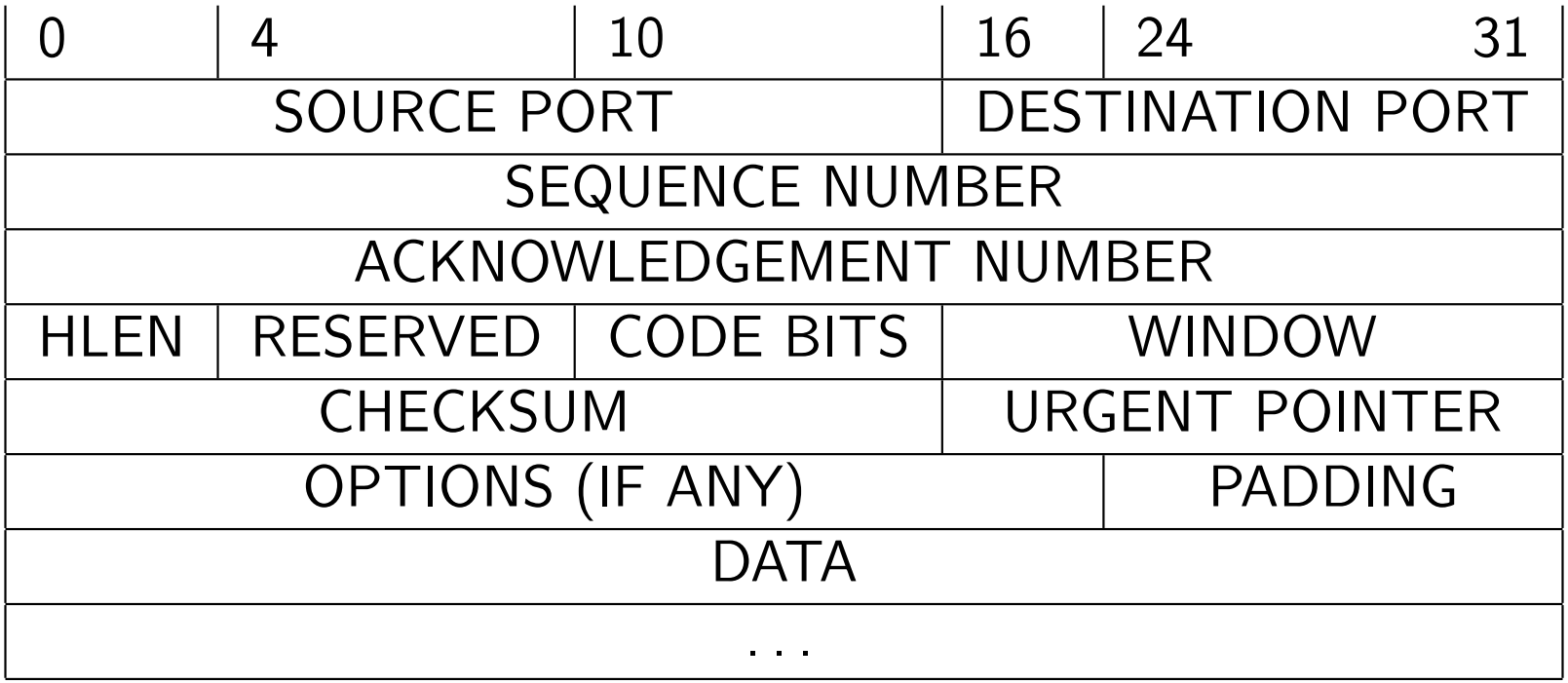


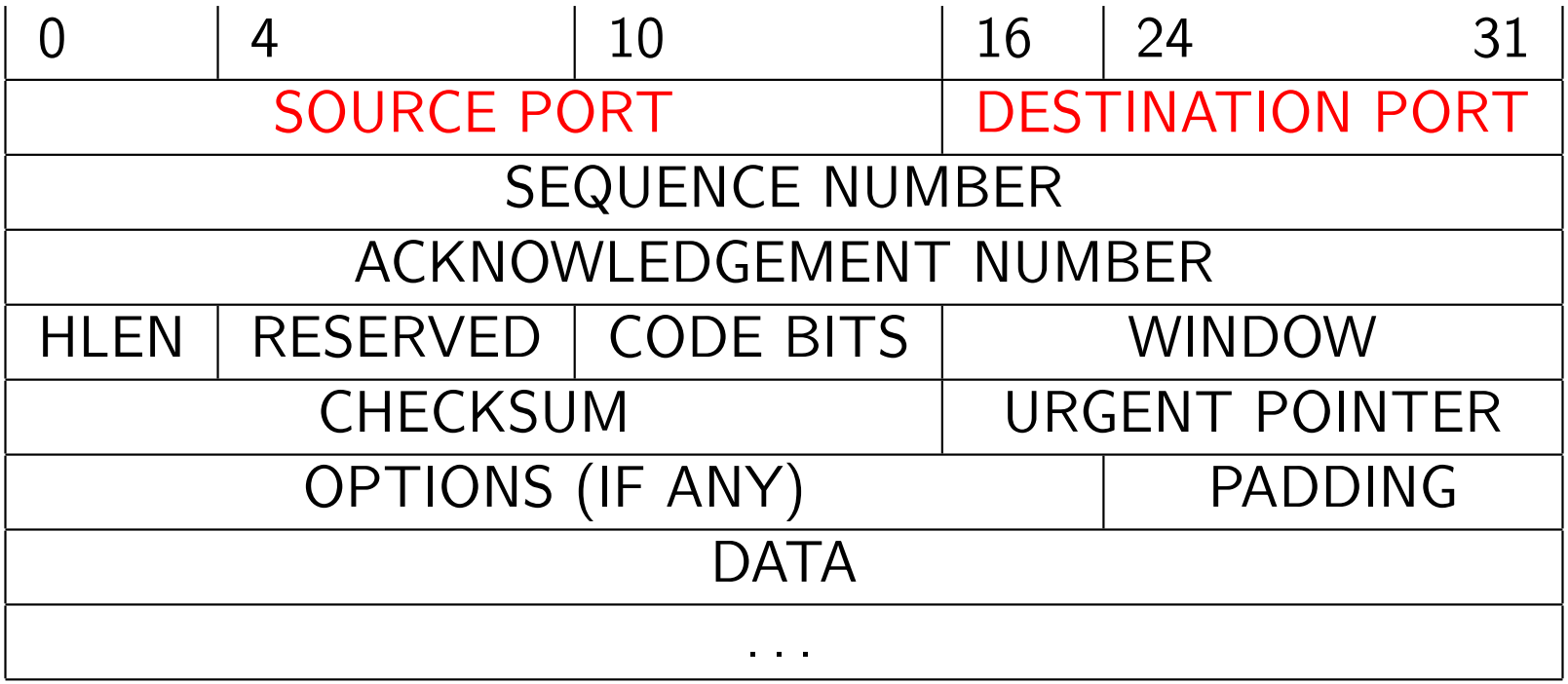
Figure 1: A TCP *listen socket* in Linux is composed of two data structures: the *request hash table*, which holds request sockets, and the *accept queue*, which holds established TCP sockets. The listen socket performs three duties: (1) tracking connection initiation requests on SYN packet reception, (2) storing TCP connections that finished the three-way handshake, and (3) supplying TCP connections to applications on calls to `accept()`.

Quelle: Aleksey Pesterev et al.: *Improving Network Connection Locality on Multicore Systems*, EuroSys'12

TCP-Pakete heißen Segmente.



TCP-Pakete heißen Segmente.



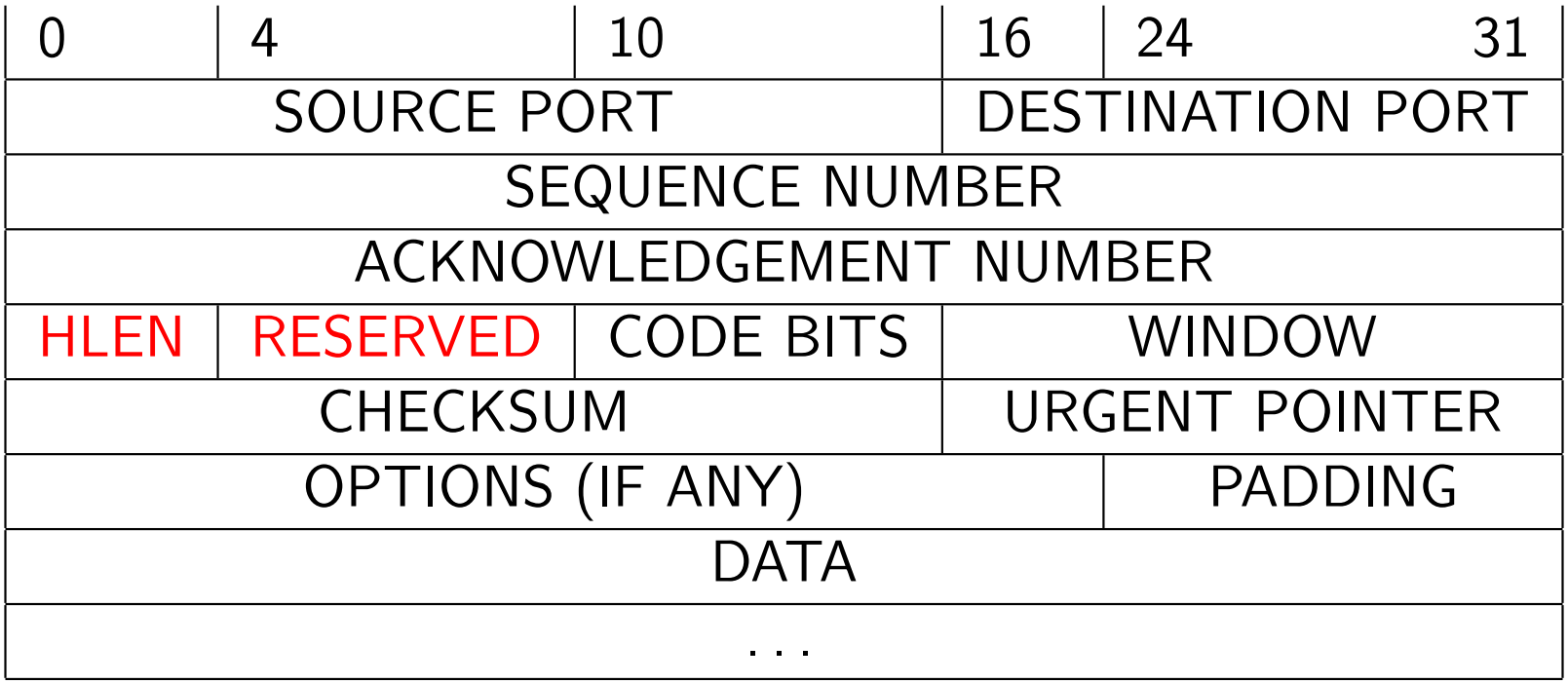
SOURCE PORT : Portnummer des Absenders
DESTINATION PORT : Portnummer des Empfängers

TCP verwaltet Verbindungen auf einem System anhand der Endpunkt-Paare:
`(my host IP-address, my TCP port)` und
`(destination host IP-address, destination TCP port)`.

$\Rightarrow$ verschiedene Verbindungen können über denselben Port laufen

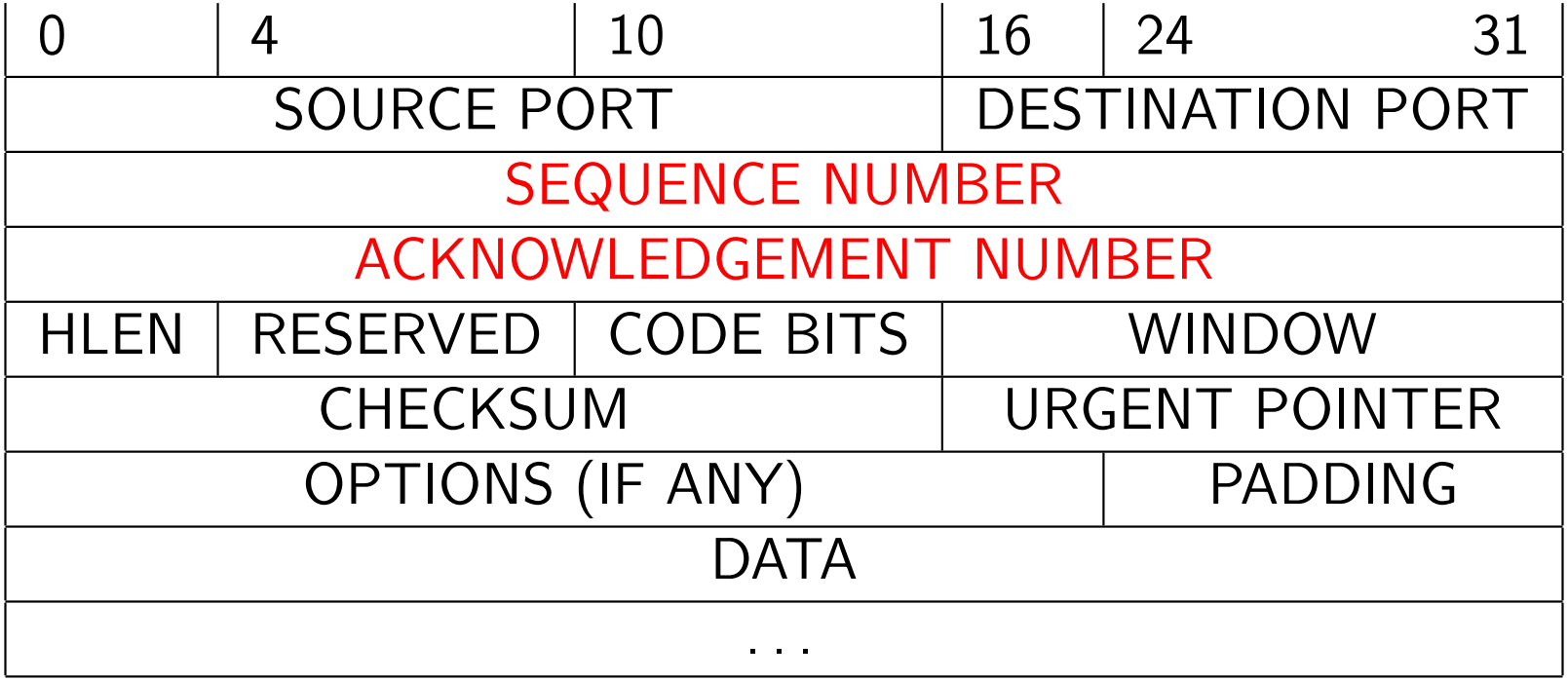
$\Longrightarrow$ Key für die Request Hash Table

TCP-Pakete heißen Segmente.



HLEN: Länge des Headers als Vielfaches von 32-Bit Wörtern
$5 \leq \text{HLEN} \leq 15 \Longrightarrow 20 \text{ Bytes} \leq \text{TCP-Header} \leq 60 \text{ Bytes}$
$\Longrightarrow$ maximal 40 Bytes Options
auch: Data Offset genannt

RESERVED: unbenutztes 6-Bit Feld

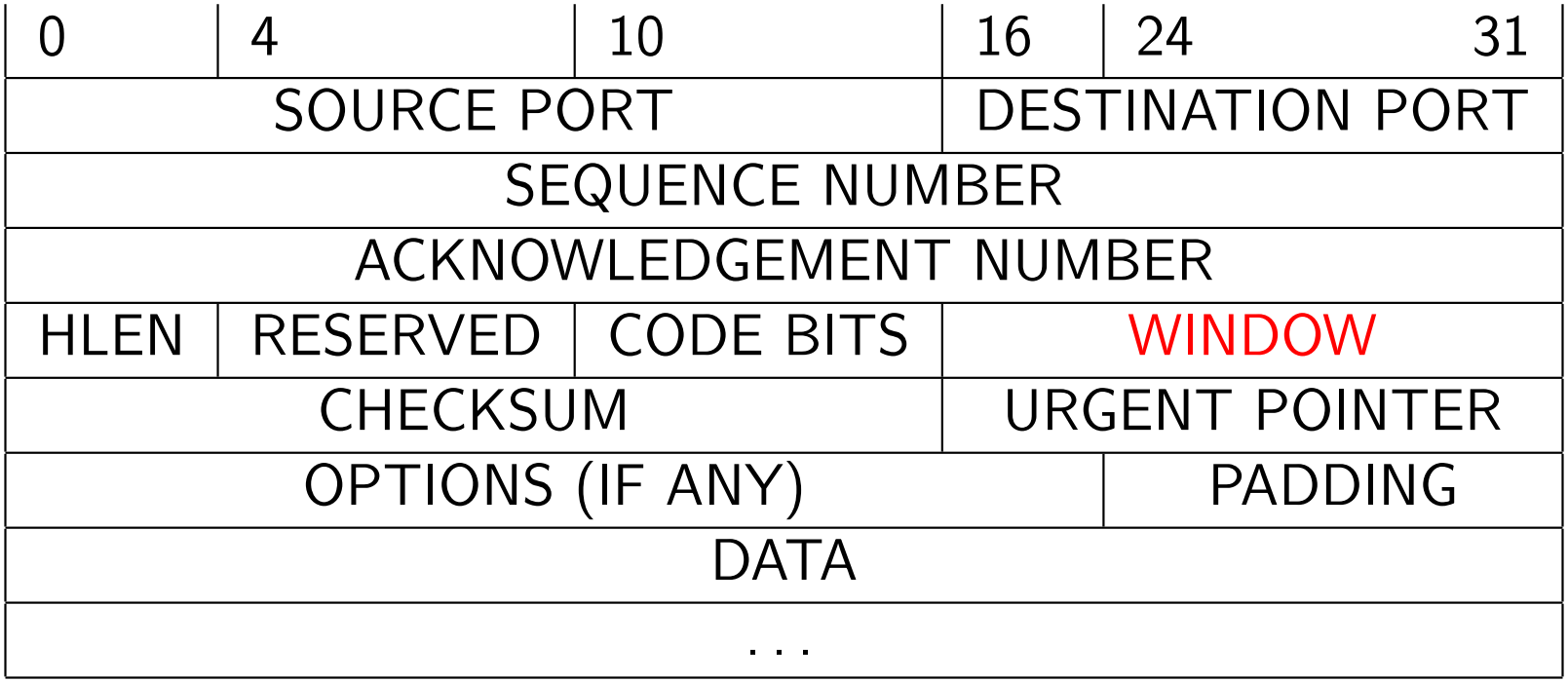


Zuverlässigkeit (Reliability)

SEQUENCE NUMBER: Nummer des ersten Byte im Data-Feld

Piggy Backing für Quittungen:
ACK. NUMBER: Nummer des nächsten erwarteten Bytes

$\Longrightarrow$ ermöglicht im Fehlerfall die effiziente Wiederholung von Daten („stream oriented")



Flusskontrolle

WINDOW: Anzahl Bytes, die ab dem letzten bestätigten Byte gesendet werden dürfen

$\Longrightarrow$ Variable-Size Sliding Window

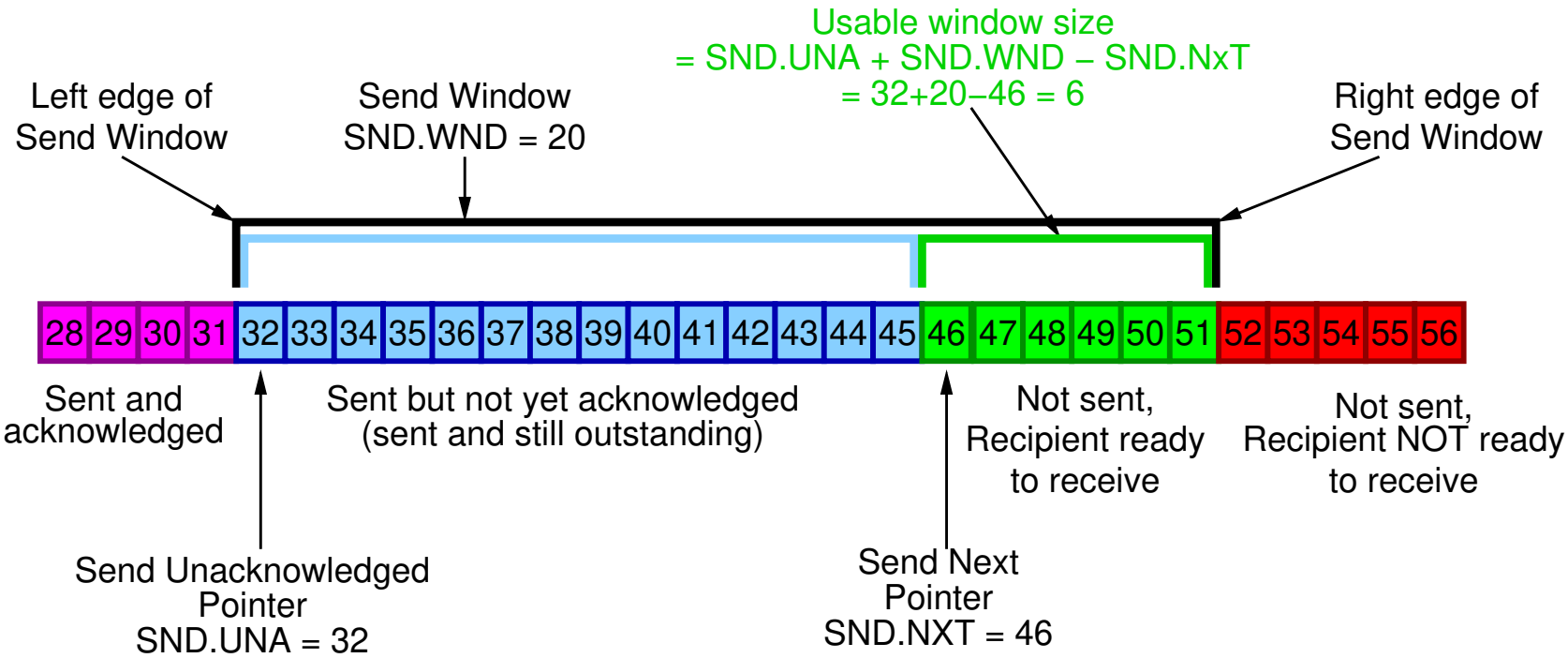
Window Scale Option siehe RFC 7323, 2014: *TCP Extensions for High Performance*

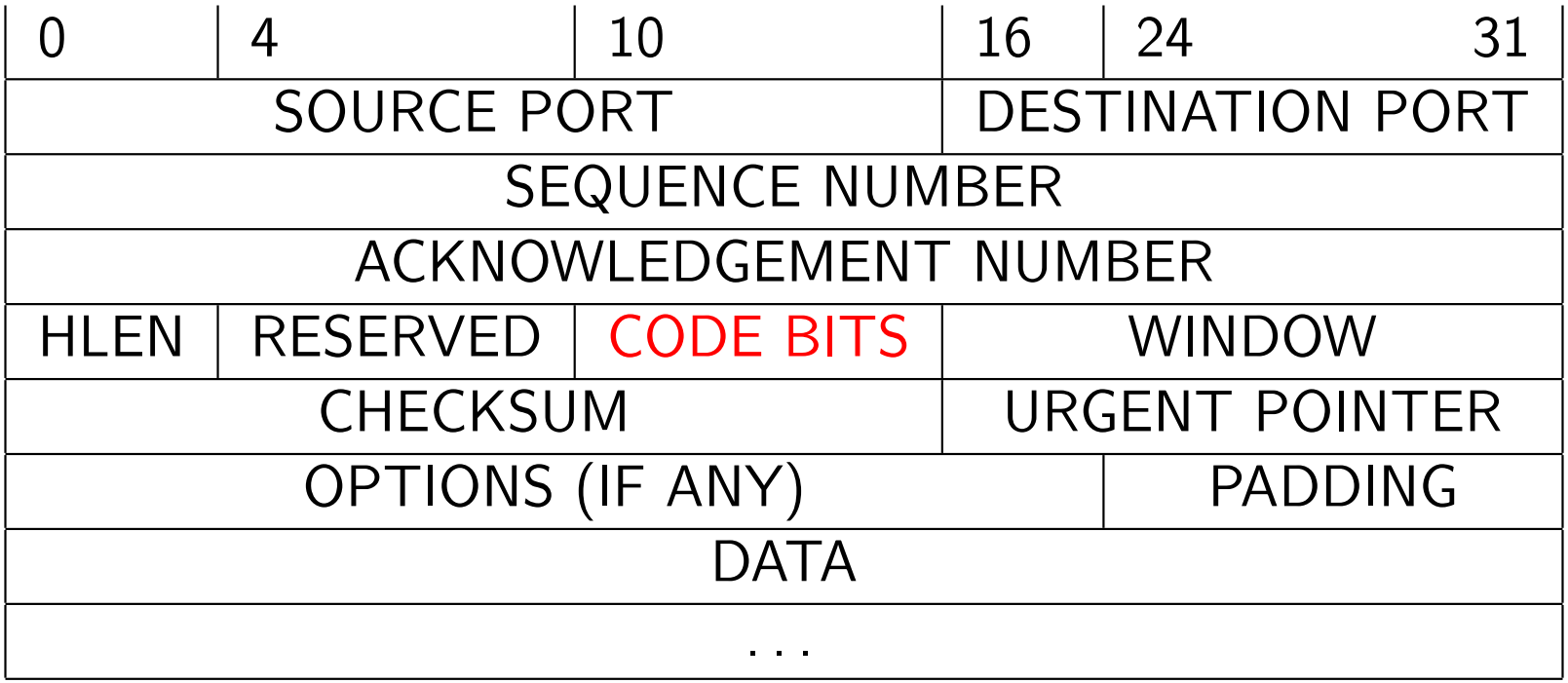
Der Transmission Control Block (TCB) ist eine Datenstruktur zur Verbindungsverwaltung. Der TCB enthält u.a.:

- local socket number
- Zeiger auf Send/Receive Buffer
- Send Sequence Variablen
- Receive Sequence Variablen

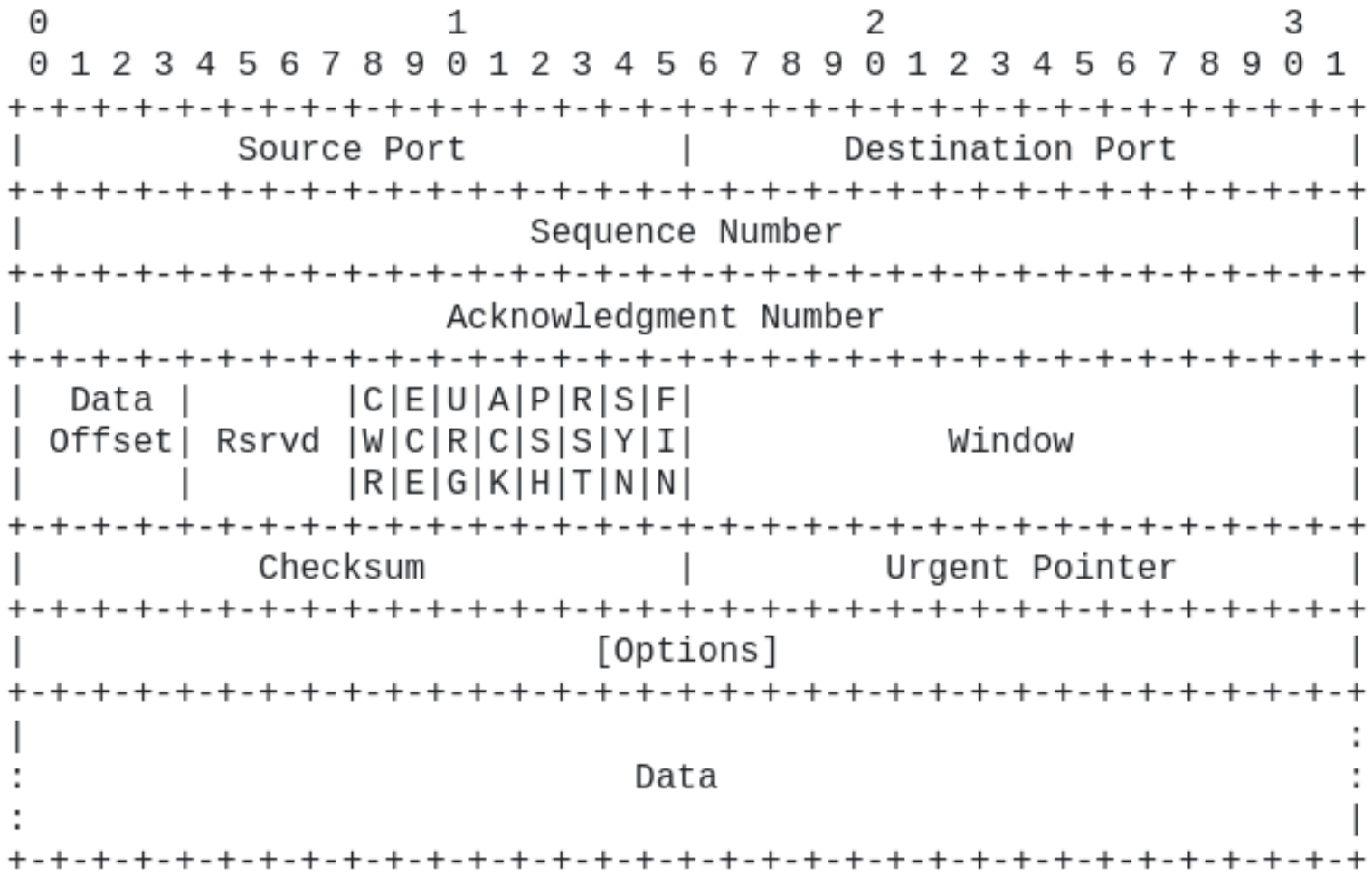
Send Sequence Variablen:

ISS:	Initial Send Sequence Number
SND.UNA:	Send unacknowledged
SND.NXT:	Send next
SND.WND:	Send window


Usable window size
= SND.UNA + SND.WND – SND.NxT
= 32+20–46 = 6
Left edge of
Send Window
Send Window
SND.WND = 20
Right edge of
Send Window
28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48 49 50 51 52 53 54 55 56
Sent and
acknowledged
Sent but not yet acknowledged
(sent and still outstanding)
Not sent,
Recipient ready
to receive
Not sent,
Recipient NOT ready
to receive
Send Unacknowledged
Pointer
SND.UNA = 32
Send Next
Pointer
SND.NXT = 46


0
4
10
16
24
31
SOURCE PORT
DESTINATION PORT
SEQUENCE NUMBER
ACKNOWLEDGEMENT NUMBER
HLEN
RESERVED
CODE BITS
WINDOW
CHECKSUM
URGENT POINTER
OPTIONS (IF ANY)
PADDING
DATA
...

Bedeutung der Bits im CODE Feld des TCP-Headers:



Note that one tick mark represents one bit position.

Figure 1: TCP Header Format

RFC 9293 (2022)

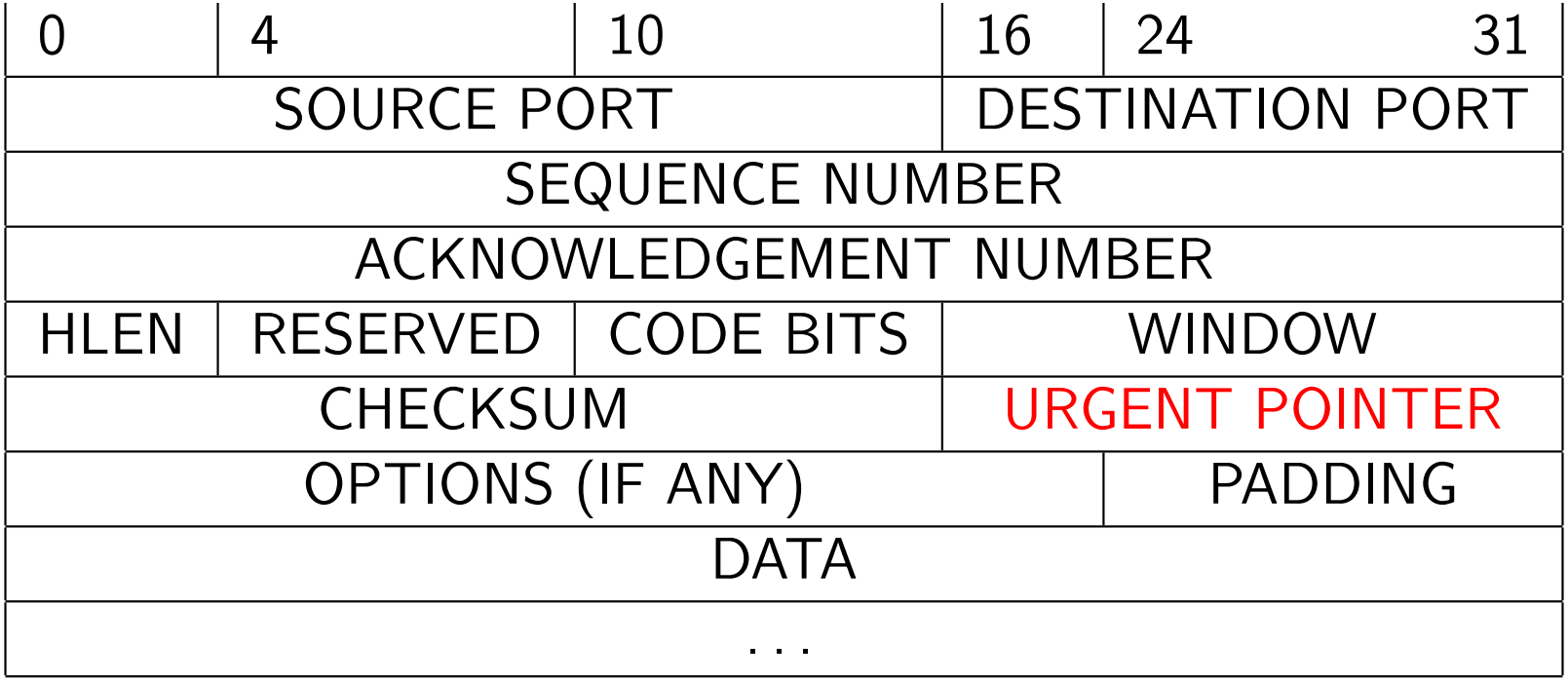
Bedeutung der Bits im CODE Feld des TCP-Headers:

Bit (left to right)	Meaning if bit set to 1
CWR	Congestion Window Reduced
ECE	Explicit Congestion Notification (ECN) Echo[18]
URG	Urgent pointer field is valid
ACK	Acknowledgement field is valid
PSH	This segment requests a push[19]
RST	Reset the connection
SYN	Synchronize sequence numbers
FIN	No more data from sender.

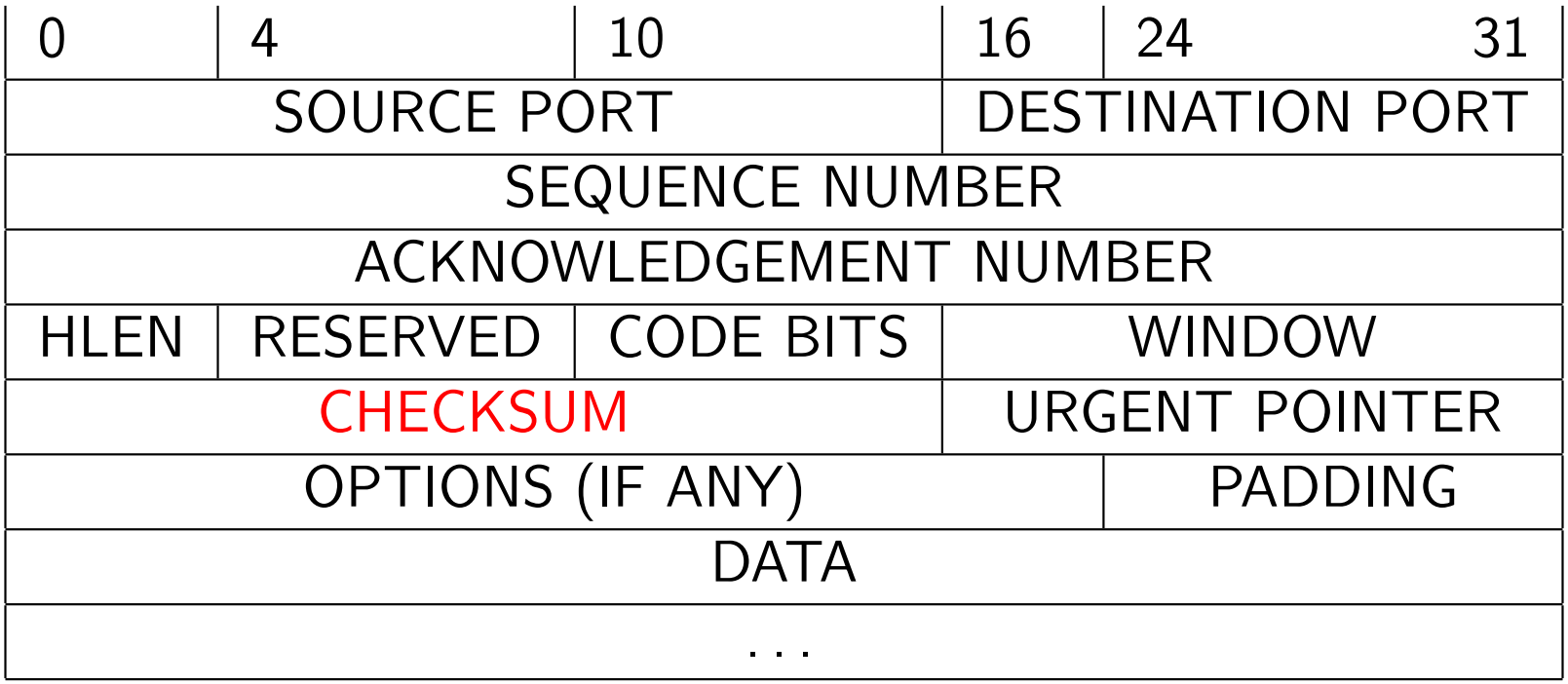
RFC 9293: As a result of implementation differences and middlebox interactions, new applications SHOULD NOT employ the TCP urgent mechanism (SHLD-13). However, TCP implementations MUST still include support for the urgent mechanism (MUST-30).

[18] RFC 3168

[19] Alles, was bisher gesendet wurde, soll dem Empfänger zugestellt werden.



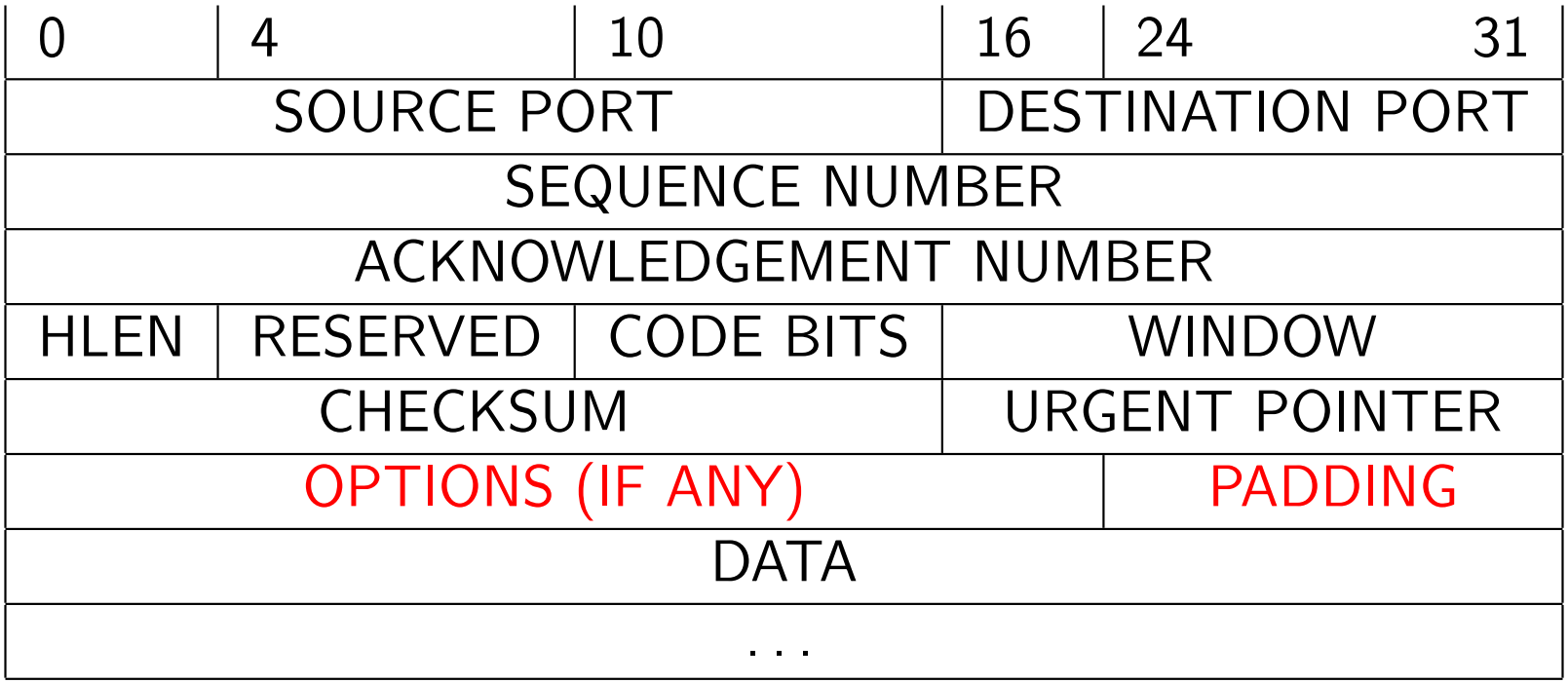
URGENT POINTER : Gibt die Position im aktuellen Segment an, an der die Urgent-Data enden



CHECKSUM: Prüfsumme über Pseudoheader gemäß Einerkomplement

Pseudoheader:

0	8	16	31
SOURCE IP ADDRESS			
DESTINATION IP ADDRESS			
00000000	PROTOCOL NUMBER	SEGMENT LENGTH (HEADER + DATA)	


0
4
10
16
24
31
SOURCE PORT
DESTINATION PORT
SEQUENCE NUMBER
ACKNOWLEDGEMENT NUMBER
HLEN
RESERVED
CODE BITS
WINDOW
CHECKSUM
URGENT POINTER
OPTIONS (IF ANY)
PADDING
DATA
...

OPTIONS

1. Maximum Segment Size Option: Aushandeln der Datengröße: Default–Wert: 536 Byte $\Longrightarrow$ ergibt Segmente der Größe $20 + 536 = 556$ (Default–Wert für IP-Paket: 576 Byte)
2. Window scale option: Ermöglicht das Erweitern der `window size` auf 2^{30} Bytes
3. Selective repeat option: Partner können aushandeln, daß `selective repeat` an Stelle von `go-back-n` benutzt wird.

Was Sie jetzt wissen sollten ...

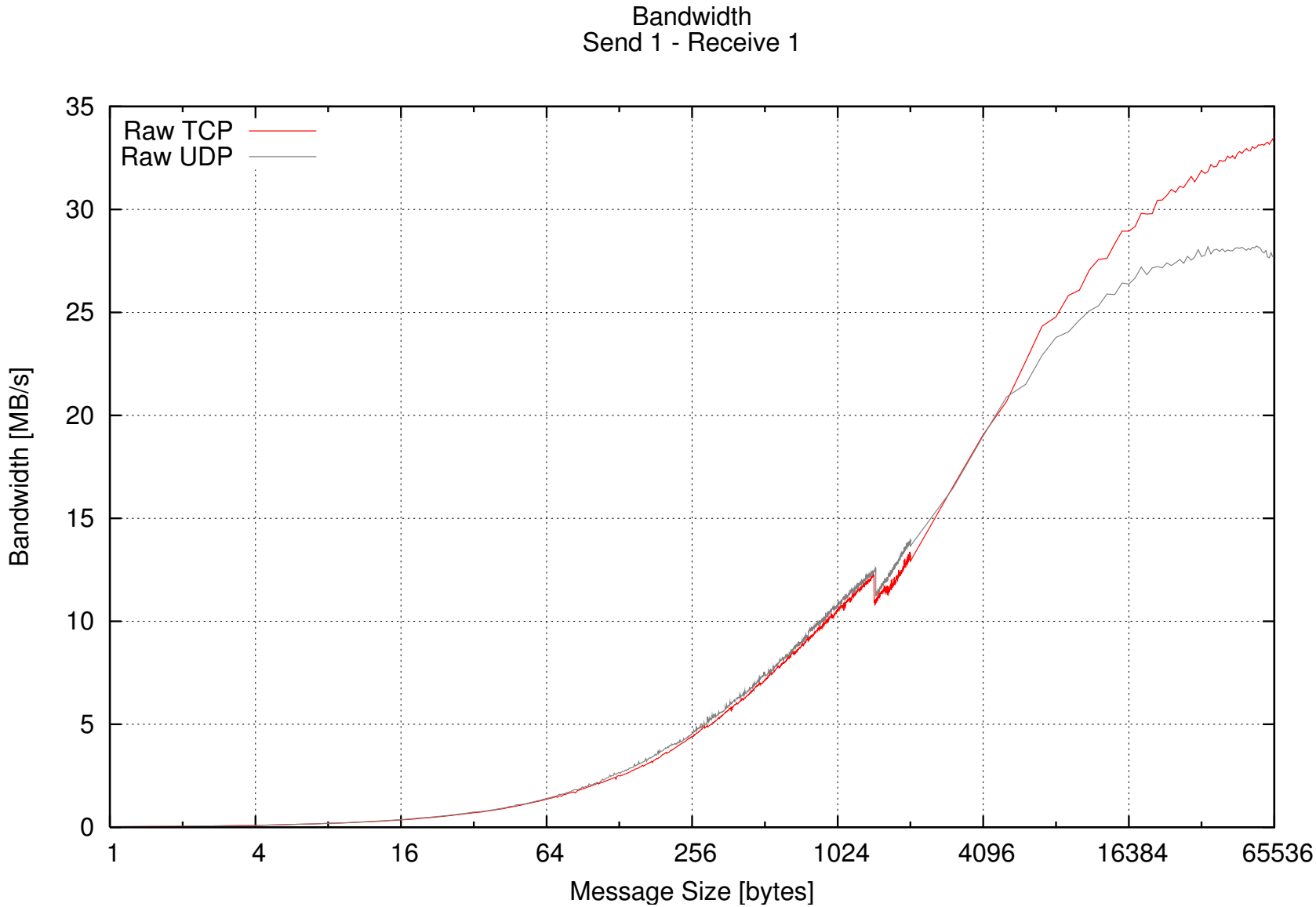
- lokale Zustellung der Pakete auf dem Zielhost anhand der Portnummer
- UDP Pseudoheader $\hat{=}$ doppelter Check der *IP*-Adresse
- UDP Pseudoheader: Implementation verletzt Schichtenmodell, d.h. die strenge Aufteilung der Funktionalitäten
- maximale UDP Datagramgröße: 65 535 Bytes
- UDP macht keine Fehlerkontrolle (keine Retransmissions)
- UDP macht keine Flusskontrolle

TCP macht für Punkt-zu-Punkt-Kommunikation (nicht für Multicast) Fehler- und Flusskontrolle (PAR-Protokoll, Variable-Size Sliding Window).
Dafür bauen TCP Sender und Empfänger mittels des 3-Way-Handshake eine virtuelle Verbindung auf.

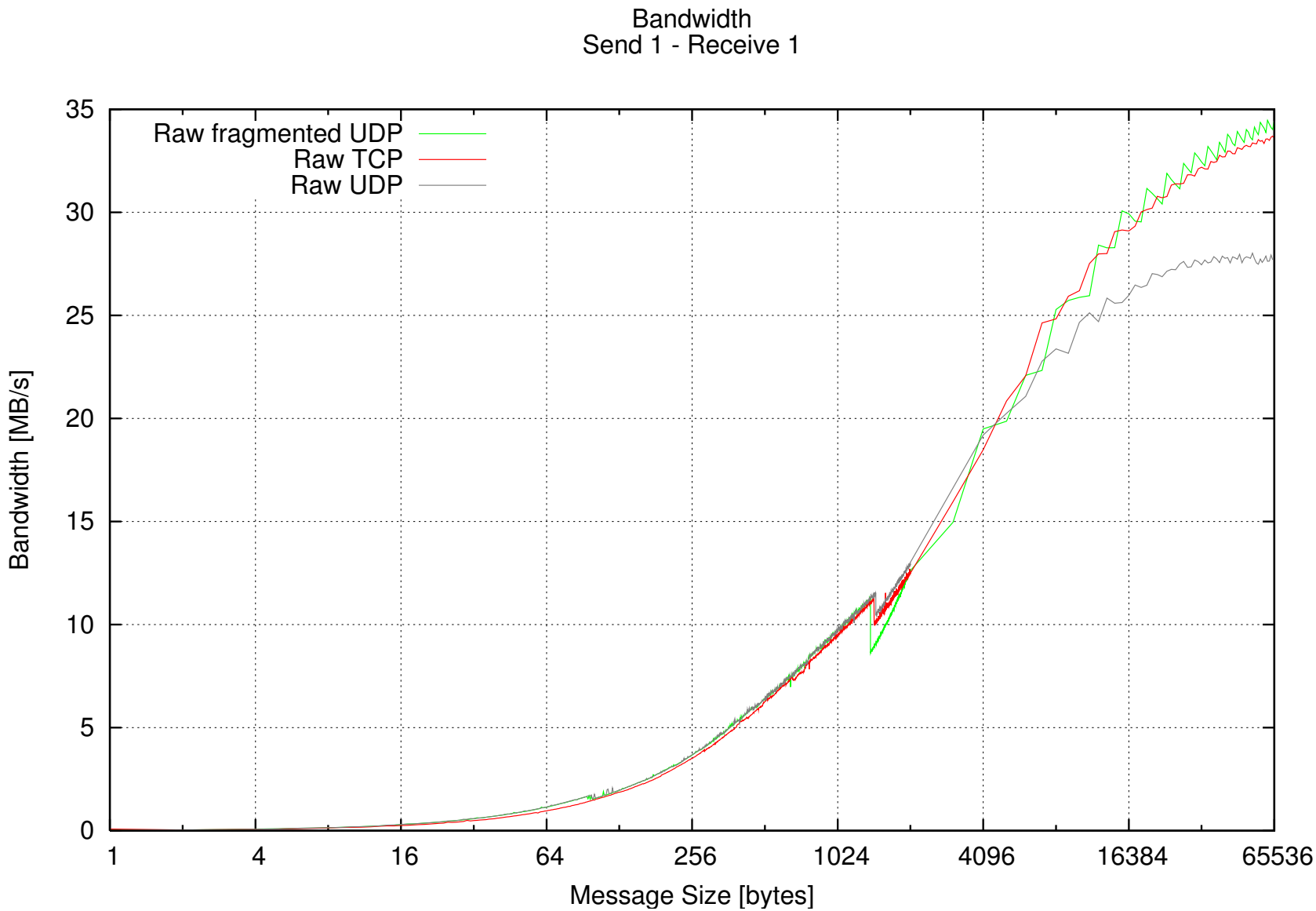
Bandbreitenvergleich TCP und UDP

Messungen von Hynek Schlawack, 2006

CPU	2x Pentium III @ 800 MHz, 256 KB Cache
Board	Gigabyte GA-6VXD C7
RAM	512 MB, 133 MHz SDRAM
PCI-Bus	32 Bit @ 66 MHz
NIC	3C996-SX (3Com Gigabit Fiber-SX Server NIC) Tigon3
	Back-to-Back



maximale UDP Datagramgröße: 65 535 Bytes


Bandwidth
Send 1 - Receive 1
Raw fragmented UDP
Raw TCP
Raw UDP
Bandwidth [MB/s]
Message Size [bytes]
0
5
10
15
20
25
30
35
1
4
16
64
256
1024
4096
16384
65536

Veranstaltungen im SoSe 2025

- Schnor/Lübke: **Konzepte Paralleler Programmierung**
  beinhaltet auch High-Performance Communication
  (InfiniBand, 100 GBit/s Ethernet)
- Schnor/Schrötter: **Sicherheit in Rechnernetzen**
  Turnus: alle 2 Jahre
- Seminar **Green Computing** (Schnor/Lübke)
- Wir vergeben bzw. vermitteln Praktikumsthemen auf Anfrage!
  aktuell: in Kooperation mit dem GFZ und PIK, sowie Themen im Bereich Security

Prüfung

Zulassung: Mit 50 % der Hausaufgabenpunkte wird man zur Klausur zugelassen.

Die Note ergibt sich aus der Klausurnote.

Klausurtermin:

Dienstag, 25.02.2025, 9:30 h – 11:30 h,
Campus Golm, Raum 2.25.F1.01, Open Book

Anmeldung: Für die Teilnahme an der Klausur ist die Anmeldung zur Modulprüfung in PULS erforderlich. Über den Anmeldungsbeginn werden Sie per E-Mail informiert.


Viel Erfolg
bei den Pruefungen!!!