Einführung in Netzwerkprogrammierung unter Perl mit Net::RawIP

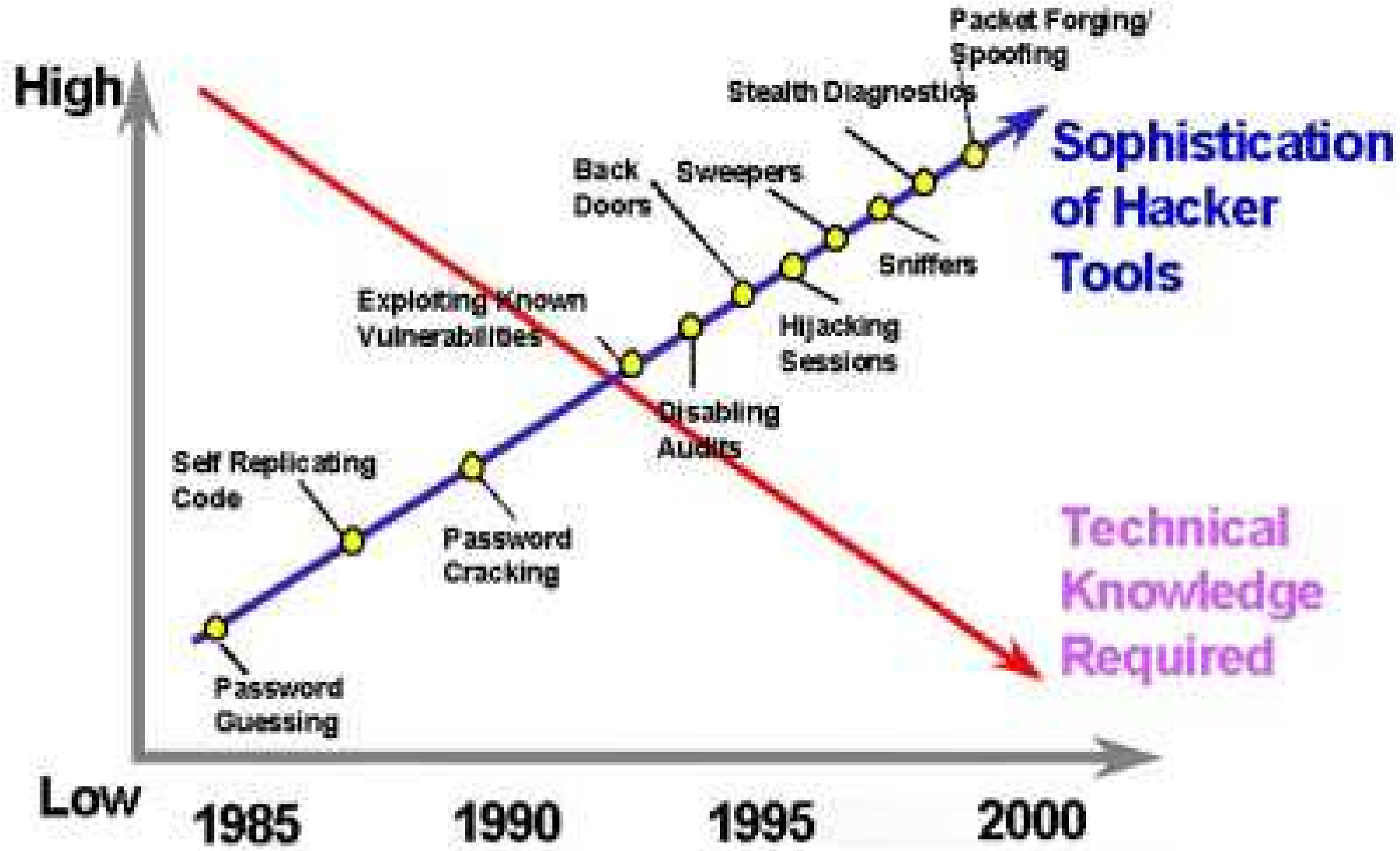
Mario: mm@koeln.ccc.de

Thorsten: tho@koeln.ccc.de

Inhalt

- Einführung in ISO/OSI-Modell
- Einführung in TCP/IP-Modell
- Netzwerkprogrammierung unter Perl
  - NetPacket::{Ethernet|ARP|ICMP|IP|TCP|UDP}
  - Net::Pcap
  - Net::RawIP
- Beispielprogramme
- Literatur

Hacker–Trends

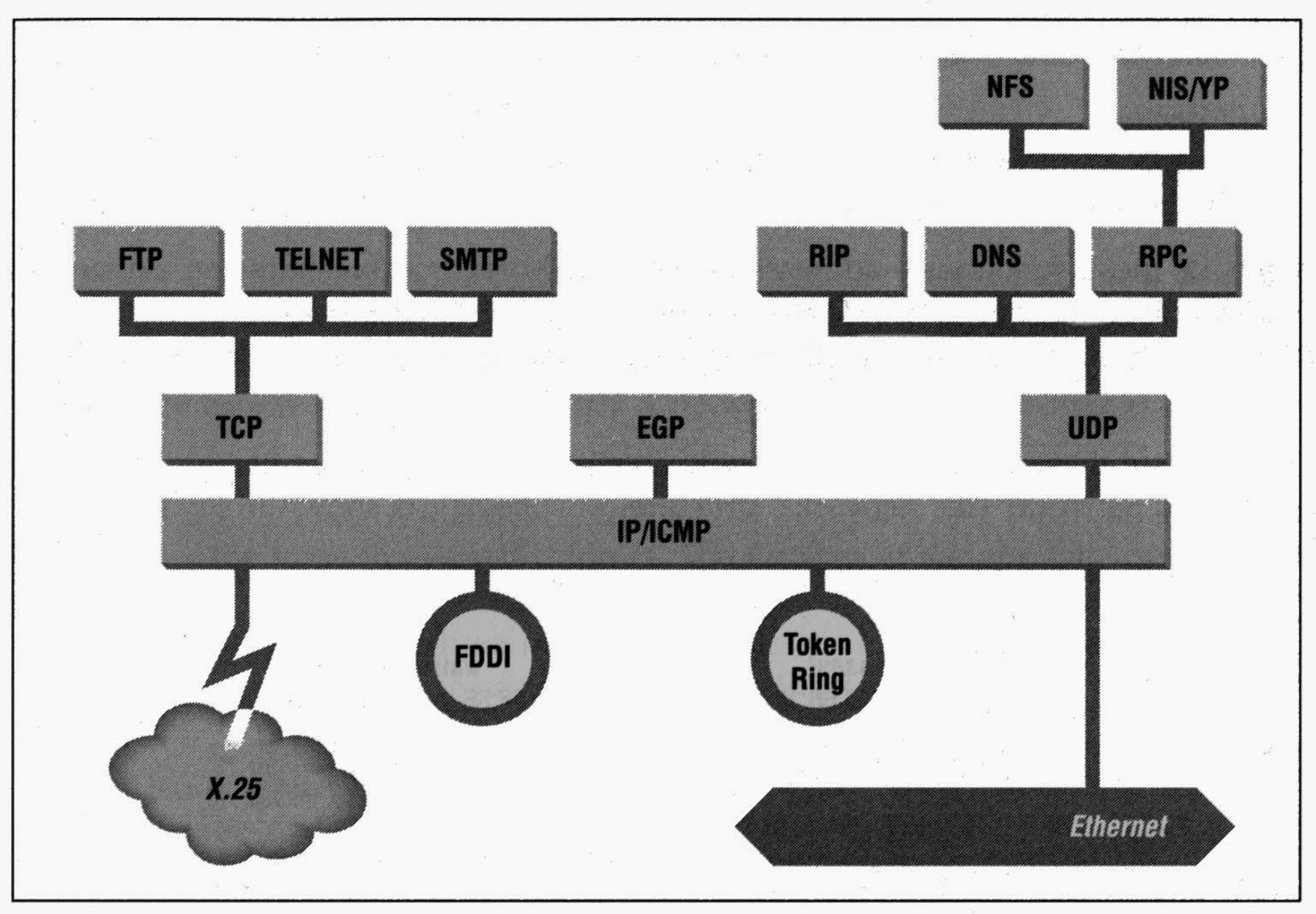


PacketForging & Perl

Was ist PacketForging und wieso Perl?

- Pakete werden auf unterster Ebene ins Netzwerk eingefügt
- Inhalt und Header der Pakete ist frei bestimmbar
- Nützlich für t00lZ und scripts
- Perl ermöglicht schnelle Entwicklung von Scripten
- Perl bietet viele Module (CPAN)

Aufbau von Netzwerken



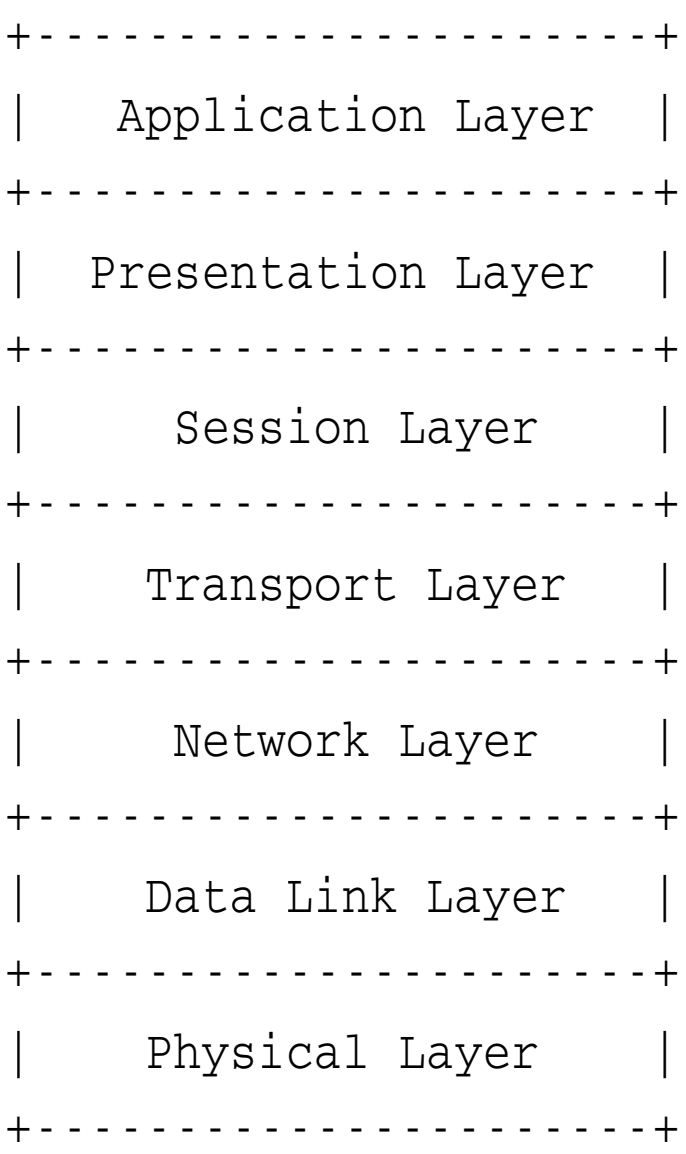
ISO/OSI-Modell

International Organisation for Standardization – ISO

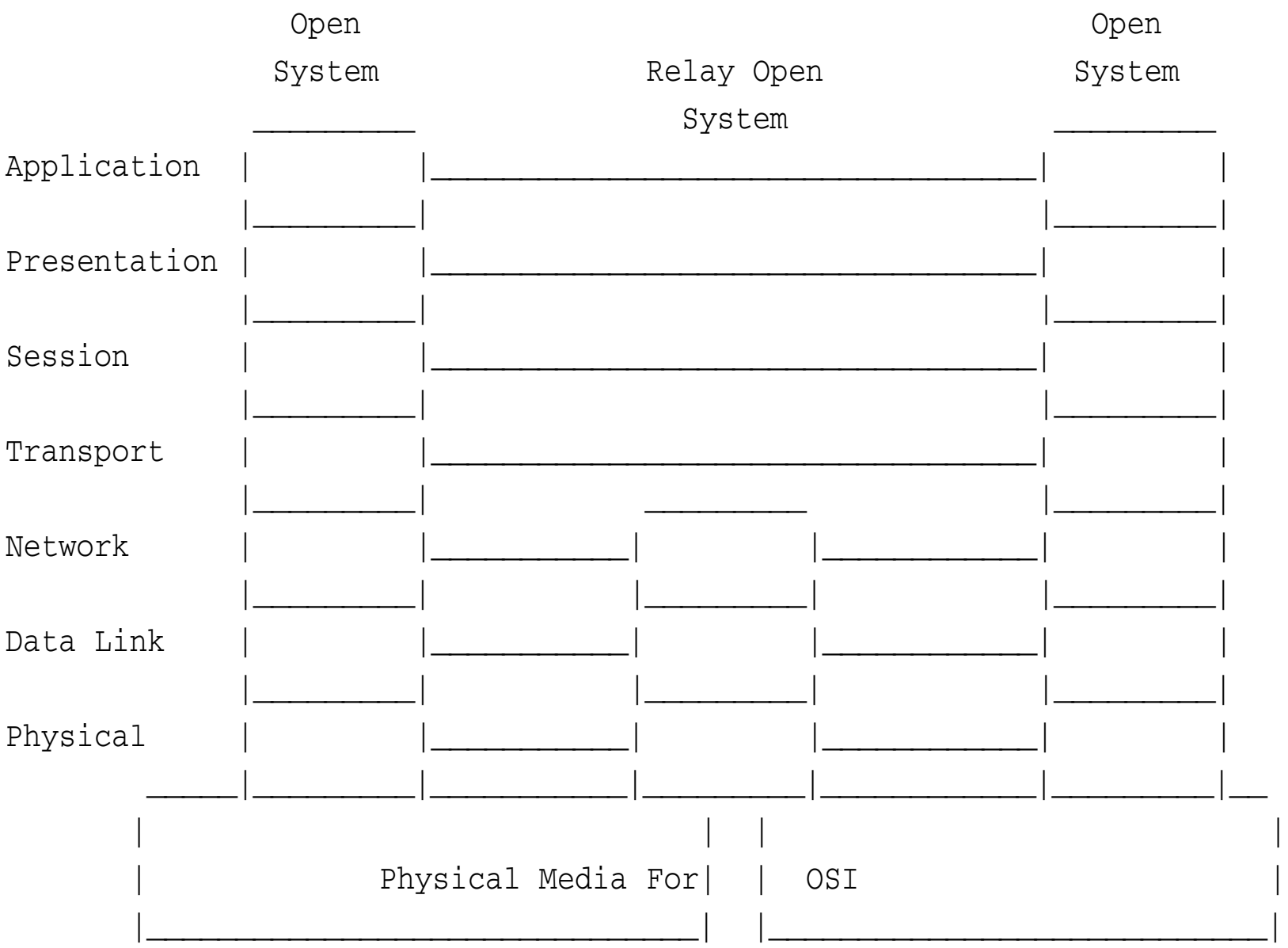
(griechisch: “isos“ = “gleich“)

- Internationaler Zusammenschluss von mehr als 140 nationalen Normierungsinstituten
- Gründung 1947, http://www.iso.ch
- “The mission of ISO is to promote the development of standardization and related activities in the world [...]“
- ISO/IEC 7498: Open System Interconnection Model (OSI)

Die sieben Schichten des OSI-Modells

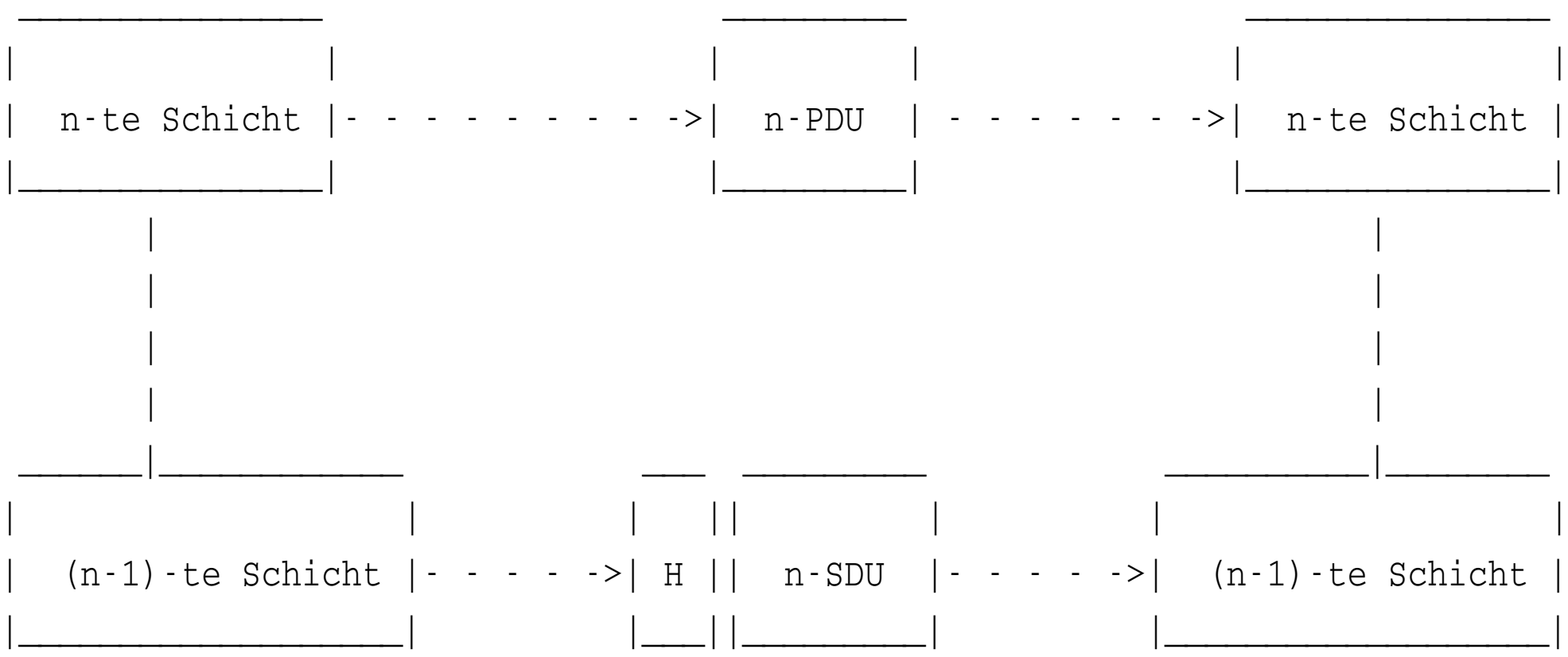


Kommunikation der verschiedenen Layer (1)



Kommunikation der verschiedenen Layer (2)

- n-te Schicht kommuniziert mit (n+1)-ter und (n-1)-ter Schicht
- Zerlegung der Nachrichten in Protocol Data Units (PDU)



Aufgaben der einzelnen Schichten (1)

- Physical Layer
  - Festlegung der physikalischen Bedingungen für die Übertragung
  - Elektrischer Pegel, Übertragungsgeschwindigkeit, Festlegung des Kabeltyps, Pinbelegung der Stecker, Übertragungsrichtung (uni– versus bidirektional) etc.
  - Beispiele: Koaxialkabel, Glasfaserkabel...
  - Geräte dieser Schicht: Repeater, Modem, Transceiver...

Aufgaben der einzelnen Schichten (2)

- Data Link Layer (Unterteilung in MAC und LLC)
  - Zusammenfassung der einzelnen Bits in frames, die dann einzeln übertragen werden
  - Sorgt für eine fehlerfreie Übertragung durch Prüfsummen oder andere Mechanismen
  - Regelung des Zugriffs auf das Übertragungsmedium (CSMA/CD bei Ethernet)
  - Beispiele: (Fast) Ethernet, Token Ring...
  - Geräte dieser Schicht: Bridges...

Aufgaben der einzelnen Schichten (3)

- Network Layer
  - Routing zwischen heterogenen Subnetzen und über größere Entfernungen
  - Aufbau, Erhalt und Abbau von Verbindungen durch ein Netzwerk
  - Wichtigstes Protokoll ist das Internet Protokoll (IP)
  - Stichwörter: Dijkstra-Algorithmus, BGP, OSPF...
  - Geräte dieser Schicht: Router, Gateways...

Aufgaben der einzelnen Schichten (4)

- Transport Layer
  - Fehlerfreie Übertragung zwischen den unterschiedlichen Systemen
  - Aufbau und Unterhalt einer virtuellen Verbindung zwischen zwei Prozessen
  - Wichtigste Protokolle sind das Transmission Control Protocol (TCP) und das User Datagram Protocol (UDP)
  - Stichwörter: Congestion Control, TCP Slow Start...

Aufgaben der einzelnen Schichten (5)

- Session Layer
  - Ähnlich zum Transport Layer Bereitstellung eines zuverlässigen Transports
  - Token–Management, Checkpoints
  - Festlegung auf Voll– oder Halbduplex
- Presentation Layer
  - Festlegung von Format, Kodierung und Syntax der Daten
  - Empfänger konvertiert wieder in eigenes Datenformat

Aufgaben der einzelnen Schichten (6)

- Application Layer

  Festlegung von drei Funktionsbereichen durch die OSI:

  - “User Element“ stellt die eigentliche Schnittstelle zwischen Anwendungsprozess und Kommunikationsdiensten dar
  - “Common Application Service Elements“ (CASE) sind definierte Funktionen für eine Vielzahl von Anwendungen
  - “Specific Application Service Elements“ (SASE) sind Funktionen für spezielle Anwendungen

Inhalt

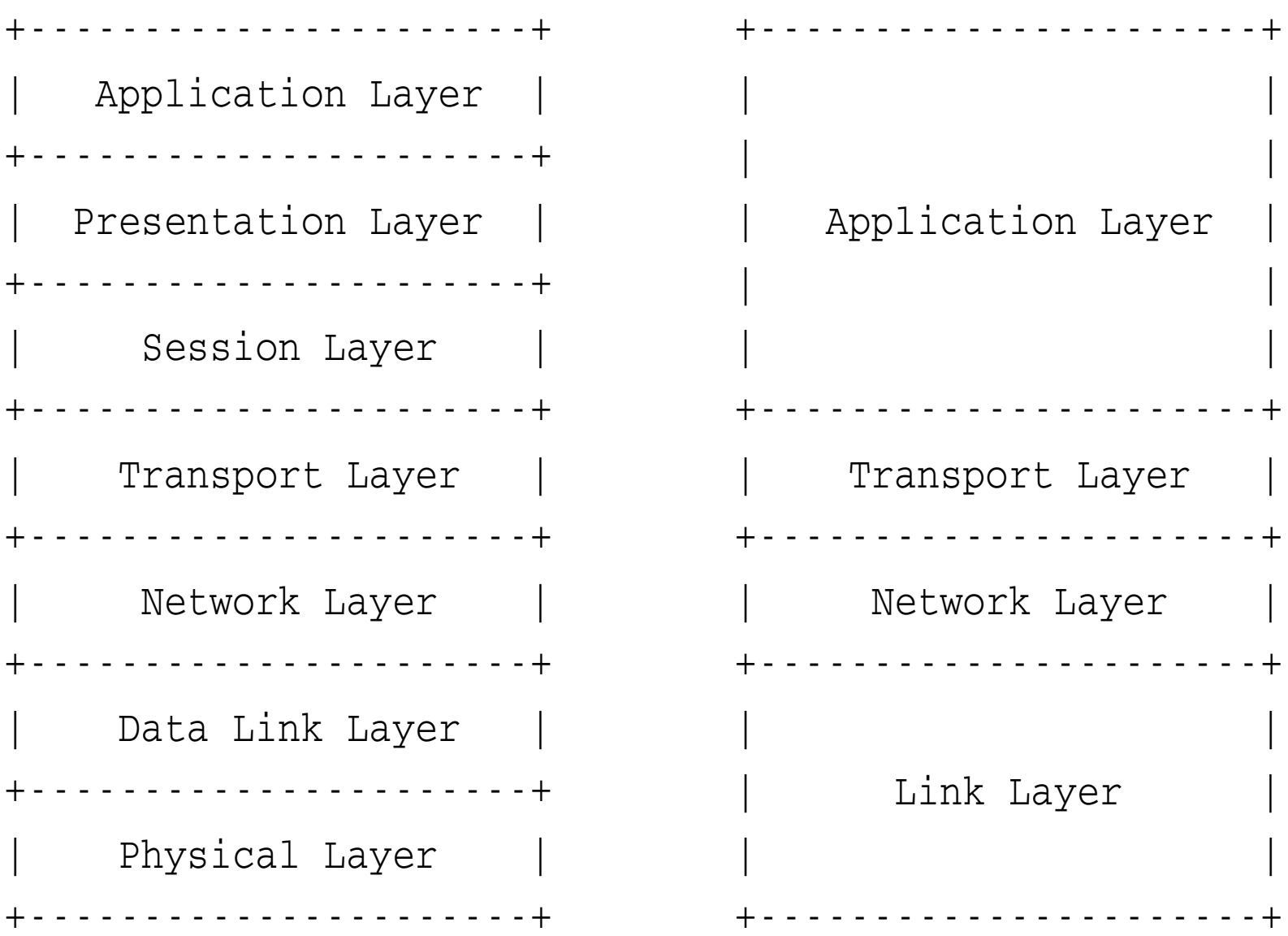
- Einführung in ISO/OSI-Modell
- Einführung in TCP/IP-Modell
- Netzwerkprogrammierung unter Perl
  - NetPacket::{Ethernet|ARP|ICMP|IP|TCP|UDP}
  - Net::Pcap
  - Net::RawIP
- Beispielprogramme
- Literatur

Probleme im OSI–Modell

- Umständliche Spezifikation
- Politisches Umfeld
- TCP/IP schon verbreitet als die OSI-Spezifikation fertig war

$\Rightarrow$ TCP/IP–Modell hat sich als Vereinfachung/Implementierung des abstrakten ISO/OSI–Modells durchgesetzt und soll im folgenden vorgestellt werden

Vergleich ISO/OSI– und TCP/IP–Modell



Geschichte von TCP/IP

1969: erste Forschung für packet-switched Netzwerke (im Gegensatz zu circuit-switched Netzwerken) durch Advanced Research Projects Agency (ARPA)

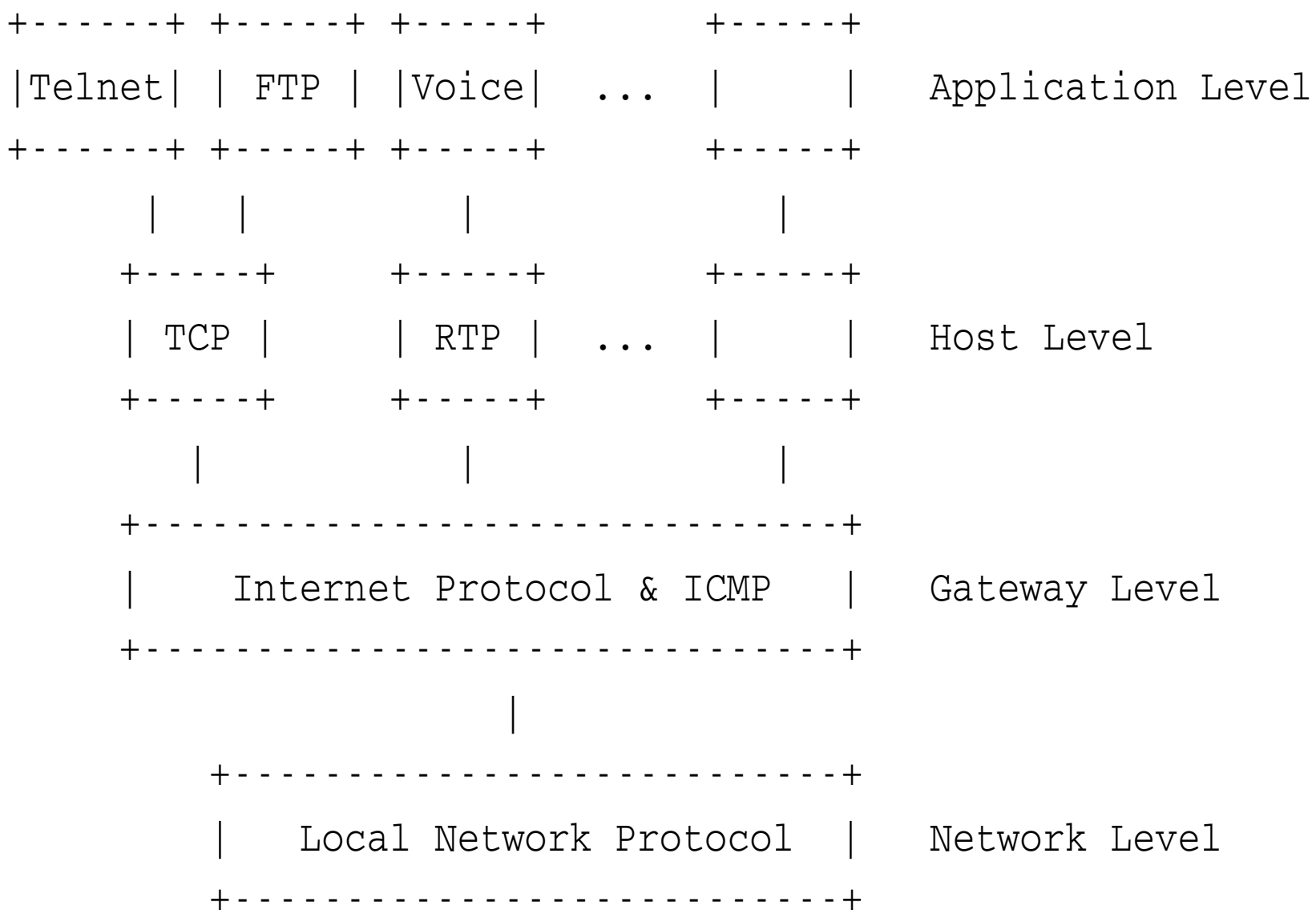
1974: Cerf und Kahn: “A Protocol for Packet Network Interconnection“

1975: DCA (Defense Communications Agency) übernimmt Kontrolle über ARPANET

1980: Berkeley UNIX (BSD 4.1) enthält TCP/IP

1983: Das ARPANET wird von NCP auf TCP/IP umgestellt

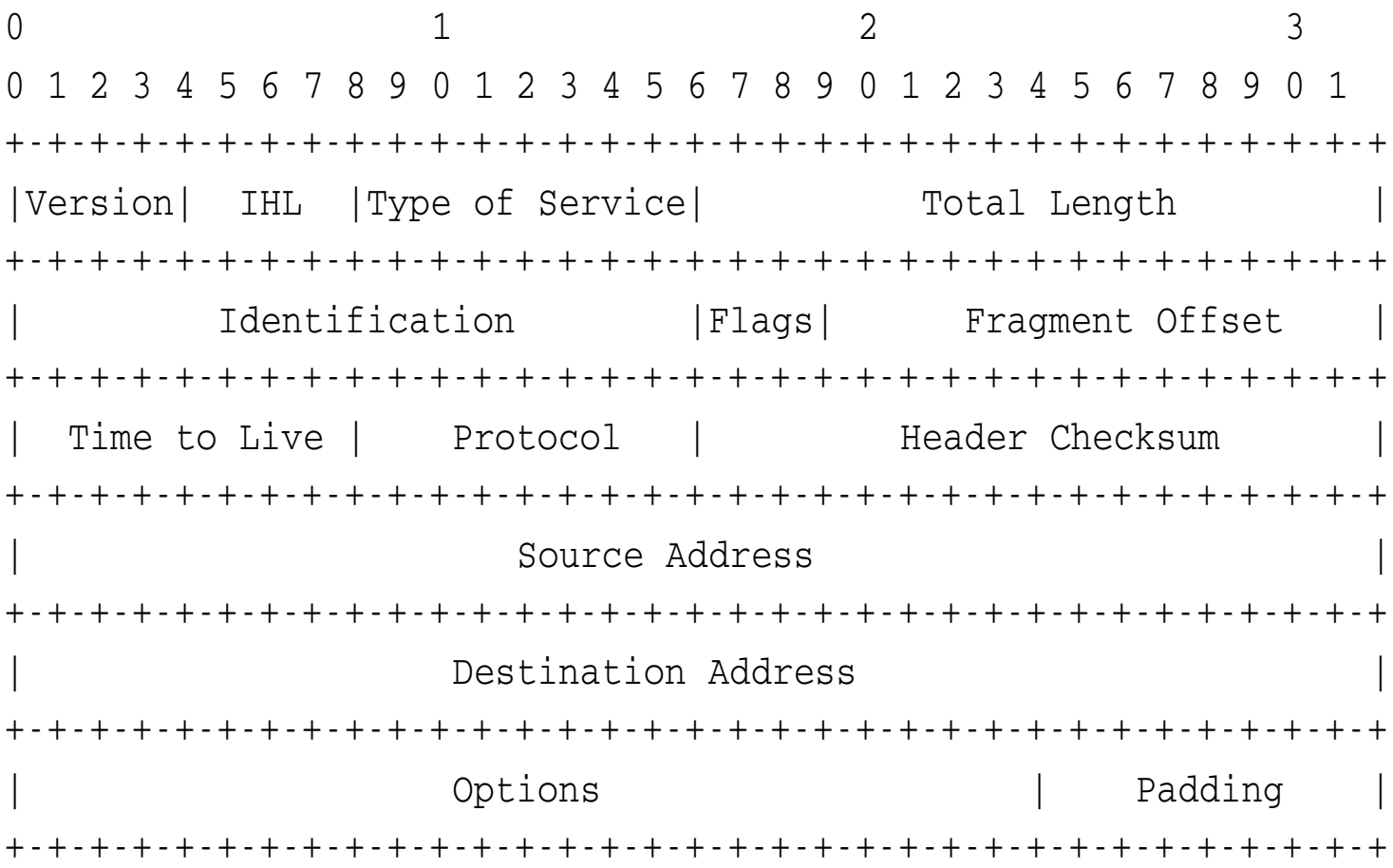
Protocol Hierarchie (RFC 793)



Internet Protocol

- RFC 791, September 1981 von Jon Postel
- Verbindungslose, unzuverlässliche Übertragung von Paketen
- Fragmentieren und Zusammensetzen der *IP–Datagramme*
- Setzt auf dem Link Layer (Ethernet, Tokenring etc.) auf
- (Ethernet-) Type value: 0x0800
- Routingentscheidungen werden von IP getroffen
- Maximale Paketgröße: 64 KB, in der Praxis meist 1500 Byte

IPv4 Header Format (RFC 791)



IPv4–Header (1)

Version	IP-Versionsnummer, hier Version 4 (4 bits)
IHL	Internet Header Length in 32 Bit Wörtern (4 bits)
ToS	Wahl der quality of service (delay, throughput und reliability), wird kaum beachtet (8 bits)
TL	Total Length (16 bits) des gesamten Datagramms
ID	Eindeutige Kennung des IP Datagramms (16 bits)

IPv4–Header (2)

Flags Verschiedene Kontroll-Flags (3 bits):

Bit 0: reserviert, muss null sein

Bit 1: (DF) 0 = May Fragment, 1 = Don’t Fragment

Bit 2: (MF) 0 = Last Fragment, 1 = More Fragments

TTL Maximale “Lebenszeit“ eines IP-Datagramms (8 bits)

TTL wird bei jedem Router um eins erniedrigt, bei 0 wird das Paket verworfen und eine ICMP-Nachricht an den Sender geschickt

IPv4–Header (3)

FOS	Offset des Fragments in 8 Byte Schritten eines Datagramms ($2^{13} = 8192$ mögliche Offsets) (13 bits)
Protocol	Angabe des im Datenteil verwendeten Protokolls
HCHK	Header Checksum, Neuberechnung an jedem Router
SA/DA	IPv4-Adresse des Senders bzw. Empfängers (32 bits)
Options	Mögliche Erweiterungen, wird kaum benutzt (variabel)
Padding	Falls nötig padding mit Nullen um Headerlänge von $n * 32$ Bit zu erreichen

IP–Adressen (1)

- Jeder host hat (mindestens) eine gültige IP–Adresse
- Darstellung der 32–Bit–Adresse als Quadrupel zu je 8 Bit, beispielsweise 212.84.209.194
- Mittels IP–Adresse und Subnet–Mask können Router das Subnet bestimmen, in dass ein Paket geroutet werden soll
- CIDR als Übergangslösung für Adress–Problem

IP–Adressen (2)

Address Allocation for Private Internets (RFC 1597):

Für private Netze wurden Adressbereiche geschaffen, die nicht über das private Netz hinaus geroutet werden

- 10.0.0.0 bis 10.255.255.255 (ein Class–A–Netz)
- 172.16.0.0 bis 172.31.255.255 (16 Class–B–Netze)
- 192.168.0.0 bis 192.168.255.255 (256 Class–C–Netze)

$\Rightarrow$ Network Address Translation nötig

Zukunft von IP

Kurzer Überlick über IPv6 (IPv5: real-time streaming protocol):

- RFC 2460, Dezember 1998 von H. Deering
- Vergrößerter Adressraum: 128 Bit-Adressen
  ($2^{128} = 340282366920938463463374607431768211456$)
- Vereinfachung des Headers – kein IHL, Protocol, Checksum
- IPSec und weitere Authentication/Privacy–Mechanismen
- Autokonfiguration von Adressen, Anycast Address

ARP/RARP (1)

Kurzer Überlick über (Reverse) Address Resolution Protocol:

- RFC 826 für ARP und RFC 903 für RARP
- IP-Adressen bilden *logische* Adressen
- Jede NIC hat (theoretisch) weltweit eindeutige MAC-Adresse (6 Byte, beispielsweise 00:FE:FE:BA:BE:00)
- Nun muss ein Mechanismus geschaffen werden, um dem Sender zu ermöglichen, IP-Datagramme im Ziel-Subnet direkt verschicken zu können

ARP/RARP (2)

- ARP-Query nur im eigenen Subnet gültig
- Benutzung der lokalen Broadcast-Adresse
- Caching der Resulate, deshalb spoofing leichter möglich

Ablauf einer ARP-Query

1. `arp who-has hub.mmweg tell vampire.mmweg`
2. `arp reply hub.mmweg is-at 0:4:76:A3:12:B4`

RARP vice versa: RARP-Server bildet MAC-Adresse auf IP-Adresse ab (heutzutage wird oft DHCP verwendet)

DHCP

Kurzer Überblick über Dynamic Host Configuration Protocol:

- RFC 2131, März 1997 von R. Droms
- Autokonfiguration der hosts: Einstellung von IP–Adresse, Default route und mehr möglich
- `DHCPDISCOVER, DHCPOFFER, DHCPREQUEST, DHCPACK/DHCPNAK, DHCPRELEASE ...`
- UDP-basiert

Später dazu noch mehr...

Überblick über ICMP (1)

Eine kurze Einführung ins Internet Control Message Protocol:

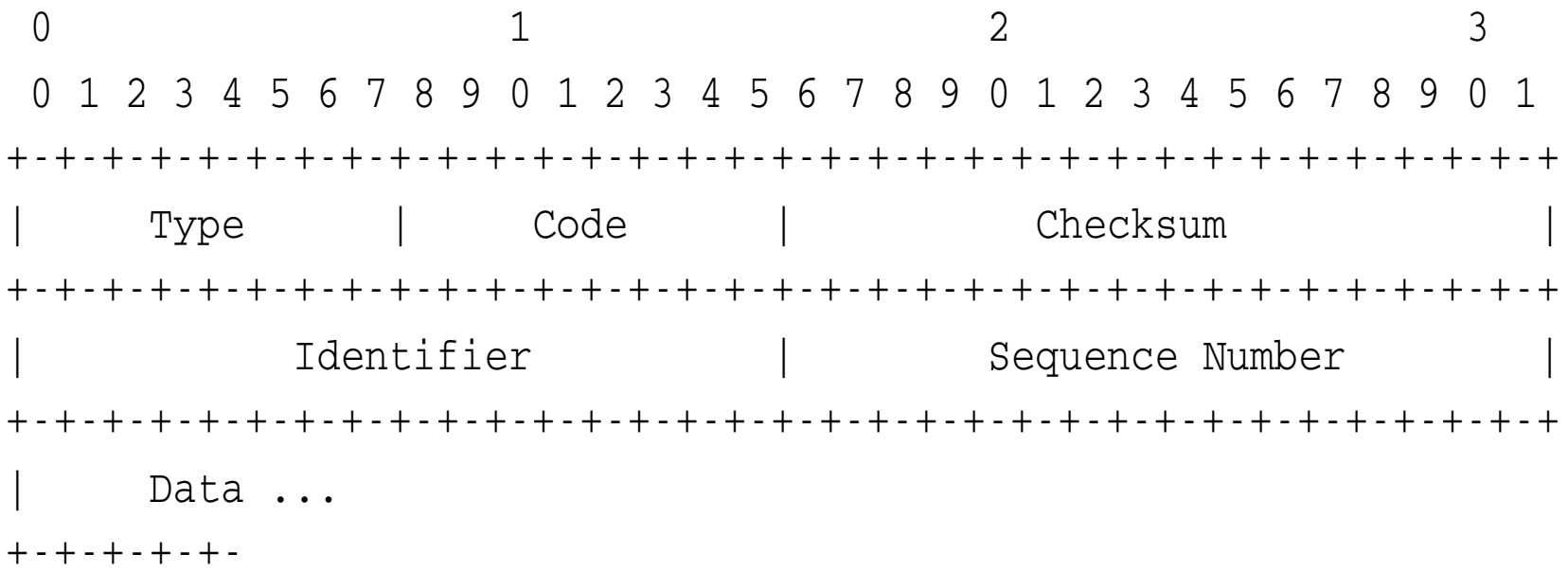
- RFC 792, September 1981 von Jon Postel
- Unzuverlässliches Protokoll, mit dessen Hilfe Fehler– und Kontrollnachrichten gesendet werden können
- Bekanntester Typ von ICMP–Nachrichten:

  Echo or Echo Reply Message (Typ 8 oder 0) – “ping“
- Weitere Typen: “Destination Unreachable“, “Time Exceeded“, “Source Quench“, “Timestamp“, ...

Überblick über ICMP (2)

- Keine ICMP-Nachrichten für fehlerhafte ICMP-Pakete
- Benutzt IP als Protokoll, ist jedoch ein integraler Bestandteil von IP (ICMP-Nachrichten benutzen den IP-Header)

Aufbau einer Echo/Echo Reply–Nachricht



Routing

Routing ist komplexes Thema, deshalb nur kurzer Überblick:

- Datagramme müssen Weg vom Sender zum Ziel allein finden
- Router setzen Routing Protokolle ein um Netzverkehr zu steuern
- Statisches versus dynamisches Routing
- RIP, OSPF (RFC 1247), BGP, EGP, ...

Vielleicht irgendwann Openchaos mit Thema “Routing“?

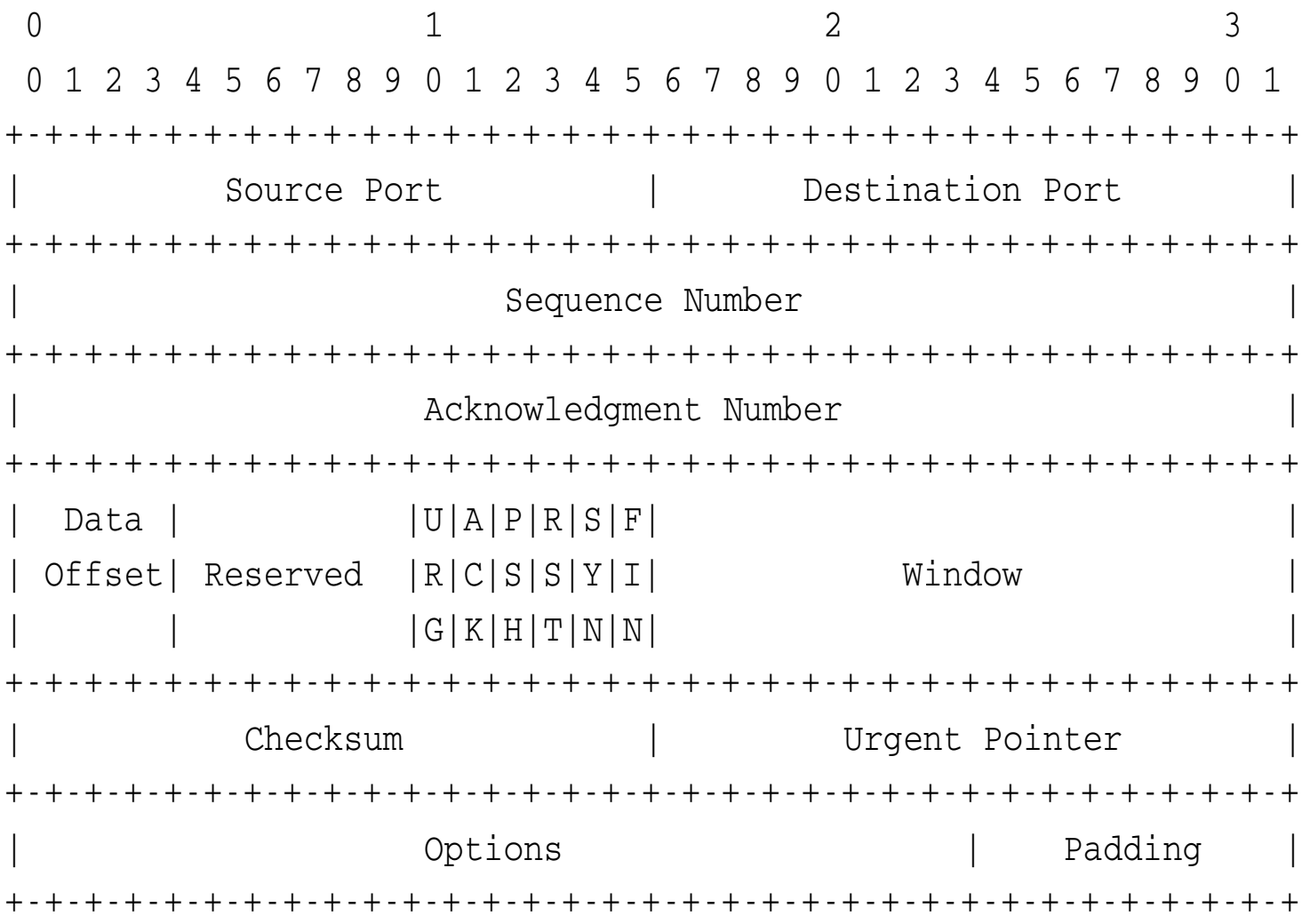
Inhalt

- Einführung in ISO/OSI-Modell
- Einführung in TCP/IP-Modell
- Netzwerkprogrammierung unter Perl
  - NetPacket::{Ethernet|ARP|ICMP|IP|TCP|UDP}
  - Net::Pcap
  - Net::RawIP
- Beispielprogramme
- Literatur

Transmission Control Protocol

- RFC 793, September 1981 von Jon Postel
- Zuverlässige, verbindungsorientierte Verbindung
- Verwendung bei allen Diensten, die zuverlässliche Verbindung voraussetzen: HTTP, FTP, ssh, ...
- bidirektionale Verbindung
- Applikation wird durch Port adressiert

TCP Header Format (RFC 793)



TCP–Header (1)

SP/DP Source/Destination–Portnummer (16 bits)

SN Sequence Number (32 bits)

Nummer des ersten Datenbytes im jeweiligen Segment

AN Acknowledgement Number (32 bits)

Falls das ACK-Bit gesetzt ist, dann enthält dieses Feld die erwartete Sequence Number des näxten Pakets

DOS Data Offset (4 bits)

Anzahl der 32 bit-Wörter im TCP-Header

R Reserved (6 bits) for future use

TCP–Header (2)

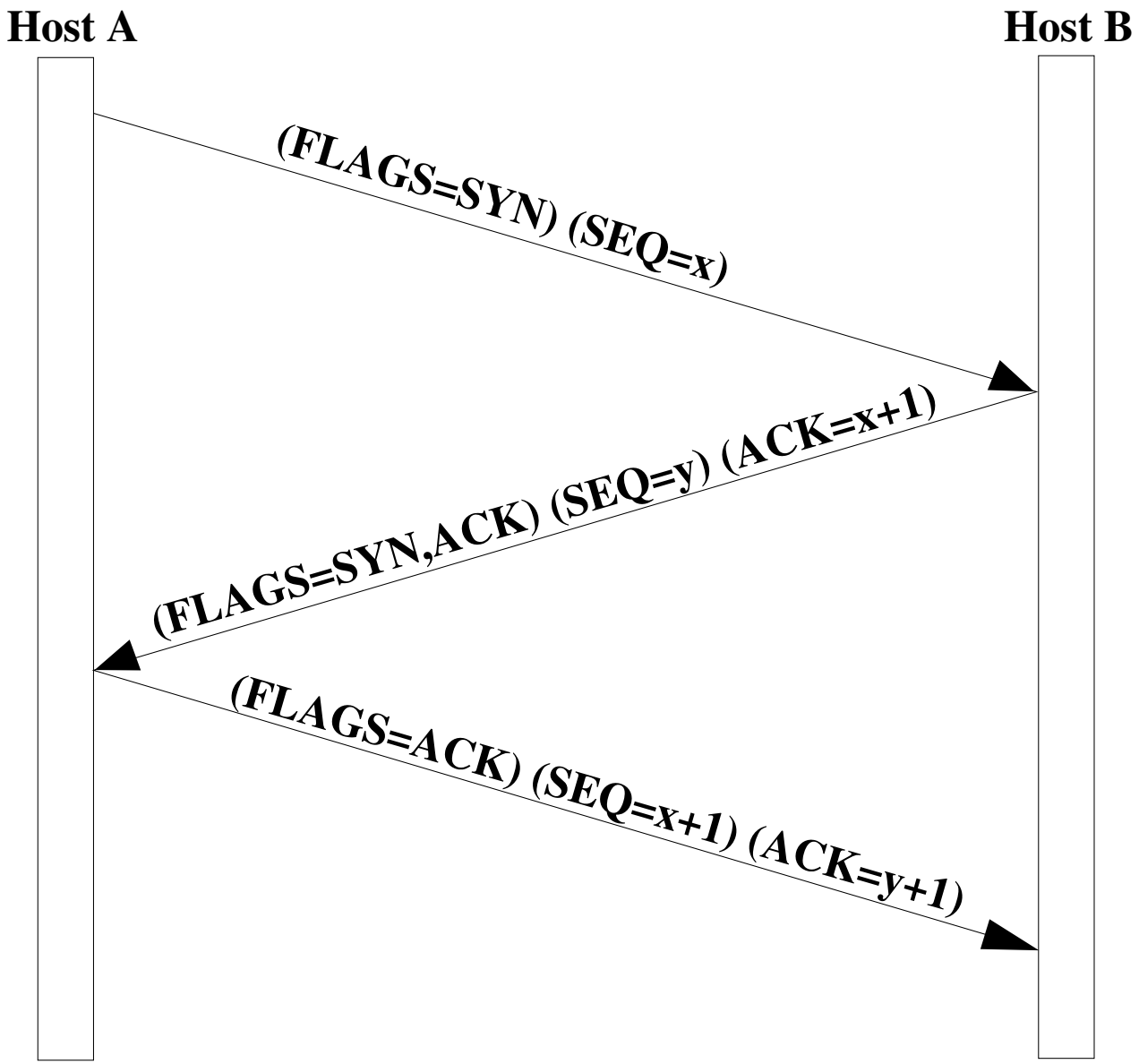
Kontroll-Bits/Flags (von links nach rechts)

URG	Urgent Pointer field significant
ACK	Acknowledgment field significant
PSH	Push Function (Daten in diesem Segment sofort der Anwendung übergeben)
RST	Reset the connection
SYN	Synchronize sequence numbers
FIN	No more data from sender

TCP–Header (3)

Window	Anzahl Bytes, die Empfänger ab dem letzten bestätigten Byte empfangen kann ohne auf ACK zu warten
CHK	Checksumme (16 bits): alle 16-Bit Wörter werden im 1-er Komplement addiert und die Summe ermittelt
URGPTR	Urgent Pointer (16 bits) ergibt zusammen mit der SN Zeiger auf wichtiges Datenbyte
Options	Optionale Funktionen, vor allem für *Maximum Segment Size* (variabel)
Padding	Null–Padding falls nötig

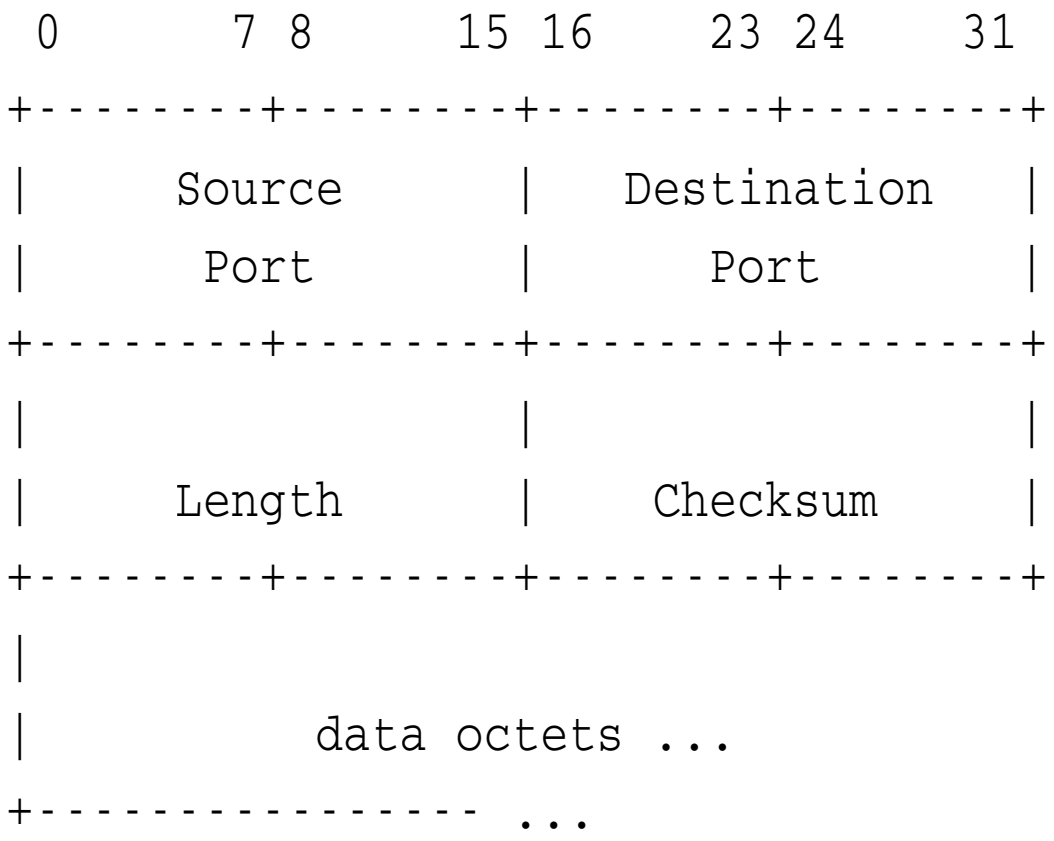
TCP Three–Way–Handshake



User Datagram Protocol

- RFC 768, August 1980 von Jon Postel
- Verbindungslose, unzuverlässliche Verbindung
- “Keep it simple“
- Verwendung vor allem bei DNS, RIP, NFS, . . .
- Nur kurzer Header (8 Byte)
- Keine Garantie für Zustellung, kein erneutes Senden nach Timeout

UDP Header Format (RFC 768)



Inhalt

- Einführung in ISO/OSI-Modell
- Einführung in TCP/IP-Modell
- Netzwerkprogrammierung unter Perl
  - NetPacket::{Ethernet|ARP|ICMP|IP|TCP|UDP}
  - Net::Pcap
  - Net::RawIP
- Beispielprogramme
- Literatur

NetPacket::* – Einführung

perldoc zu NetPacket:

“NetPacket“ – modules to assemble/disassemble network packets at the protocol level.

- Decodieren: $obj = NetPacket::* –> decode (rawpacket)
- Zugriff auf Headerfield: $obj –> {headerfield}
- Verarbeitet Pakete von Net::Pcap
- Pakete werden von der niedrigsten zur höchsten Schicht decodiert

NetPacket::IP – Adress-Sniffer

```
#!/usr/bin/perl -w

use strict;
use Net::PcapUtils;
use NetPacket::Ethernet qw(:strip);
use NetPacket::IP;

sub process_pkt {
    my ($user, $hdr, $pkt) = @_;
    #pakete werden von der niedrigsten zur hoechsten schicht decodiert
    my $ip_obj = NetPacket::IP->decode(eth_strip($pkt));
    print("$ip_obj->{src_ip}:$ip_obj->{dest_ip} $ip_obj->{proto}\n");
}

Net::PcapUtils::loop(\&process_pkt, FILTER => 'ip');
```

NetPacket::TCP – Sniffer (1)

```
#!/usr/bin/perl -w

use strict;

use Net::PcapUtils;
use NetPacket::Ethernet qw(:strip);
use NetPacket::IP qw(:strip);
use NetPacket::TCP;

my($device) = @ARGV or die "give me an interface plz.";
$|=0;

# netpcaputils simple hui
Net::PcapUtils::loop(\&callback);
```

NetPacket::TCP – Sniffer (2)

```
sub callback {
            my ($arg, $hdr, $pkt) = @_;
            $eth_obj = NetPacket::Ethernet->decode($pkt);
            $ip_obj  = NetPacket::IP->decode($eth_obj->{data});
            $tcp_obj = NetPacket::TCP->decode($ip_obj->{data});
            printf "%-15s:%-5s => %-15s:%-5s\n",
                $ip_obj->{src_ip}, $tcp_obj->{src_port},
                $ip_obj->{dest_ip}, $tcp_obj->{dest_port};
}
```

Net::Pcap – Einführung

perldoc zu Net::Pcap:

“A system-independent interface for user-level packet capture.“

Perl-bindings für `pcap(3)`

Auswahl an bereitgestellten Funktionen:

- open_live: liest Daten vom Netzwerkdevice
- loop: ruft callback–Funktion auf
- close: beendet den capture–Vorgang

Net::Pcap – Sniffer

```
#!/usr/bin/perl -w

use strict;
use Net::Pcap;

my $snaplen=1024; my $promisc=1; my $to_ms=30; my $count=-1;

$object = Net::Pcap::open_live($device, $snaplen, $promisc, $to_ms, \$err);

Net::Pcap::loop($object, $count, \&callback, $arg);
Net::Pcap::close($object);

sub callback {
           my ($arg, $hdr, $pkt) = @_;
           print "packet captured";
}
```

Net::RawIP – Einführung

perldoc zu Net::RawIP:

“Perl extension for manipulate raw ip packets with interface to libpcap“

Auswahl an bereitgestellten Funktionen:

- set: setzt headerfields und payload
- send: sendet Paket
- pcapinit: filtert unrelevante Pakete raus

Net::RawIP – einfaches Paket

```
#!/usr/bin/perl -w

use strict;
use Net::RawIP;

my $pkt = new Net::RawIP ({tcp =>{}});

$pkt->set({ip  => { saddr => 'home.example',
                    daddr => 'www.example',
                    ttl   => 64},
           tcp => { source => 31337,
                    dest   => 80,
                    syn    => 1}
});

$pkt->send;
```

Net::RawIP – ping(1)

```
#!/usr/bin/perl -w

use strict;
use Net::RawIP;

my $srcip ='10.1.1.100';
my $dstip ='10.1.1.5';
($dstip) = @ARGV if @ARGV;

my $pkt = new Net::RawIP ({icmp =>{}});
$pkt->set({ip  => { saddr    => $srcip,
                    daddr    => $dstip,
                    protocol => 1,
                    tos      => 0,
                    frag_off => 0},
```

Net::RawIP – ping(2)

```
            icmp => { type     => 8,
                      code     => 44,
                      check    => 55,
                      gateway  => 66,
                      id       => 77,
                      sequence => 88,
                      unused   => 99,
                      mtu      => 101,
                      data     => 102}
});

for (my $i=1; $i < 20; $i++)  {
            $pkt->send;
}
```

Net::RawIP

Net::RawIP benutzt in seinen C Bibliotheken:

- `IPv4` weil `AF_INET`
- `SOCK_DGRAM`

  Supports datagrams (connectionless, unreliable messages of a fixed maximum length).

- `SOCK_RAW`

  Provides raw network protocol access.

Sockets

Verwendung von Sockets unter Perl

- Einfachste Art netzwerkfähige Programme zu schreiben
- Werden fast wie Dateizugriffe programmiert
- listen / connect sockets
- src ip spoofing nicht möglich (Ausnahme: raw sockets)
- Broadcast Zieladdresse möglich

Sockets

```
use IO::Socket;
my $remote_host = '172.23.23.81';
my $remote_port = '1111';
$socket = IO::Socket::INET->new(PeerAddr => $remote_host,
                                PeerPort => $remote_port,
                                Proto    => "tcp",
                                Type     => SOCK_STREAM)
    or die "Couldn't connect to $remote_host:$remote_port : $@\n";

#LocalAddr => '172.23.23.23', # => Cannot assign requested address

print $socket "PORT 23\n";
print $socket "SCAN koeln.ccc.de\n"; print $socket "QUIT\n";

while ($answer = <$socket>) { print $answer; }
close($socket);
```

PacketForging

Bewertung:

- Module wie Net::DHCP::Packet geben einfachen Zugriff auf komplexe Protokolle
- DoS, Probing, Spoofing und Injection möglich
- Komplexe Paketfolgen aufwendig zu programmieren

Inhalt

- Einführung in ISO/OSI-Modell
- Einführung in TCP/IP-Modell
- Netzwerkprogrammierung unter Perl
  - NetPacket::{Ethernet|ARP|ICMP|IP|TCP|UDP}
  - Net::Pcap
  - Net::RawIP
- **Beispielprogramme**
- Literatur

Was mensch alles mit Perl machen kann

Einige kleinere scripte von uns für euch:

- DHCP–leecher

  Belegt alle IP-Adressen in Netzwerken mit DHCP

- Gameserver–DoS

  DoS-tool unter Zuhilfenahme von Gameservern

Zu finden auf http://koeln.ccc.de

Inhalt

- Einführung in ISO/OSI-Modell
- Einführung in TCP/IP-Modell
- Netzwerkprogrammierung unter Perl
  - NetPacket::{Ethernet|ARP|ICMP|IP|TCP|UDP}
  - Net::Pcap
  - Net::RawIP
- Beispielprogramme
- Literatur

Literaturüberblick

- RFCs 791-793 etc. (http://www.rfc-editor.org)
- Comer, Douglas: Internetworking with TCP/IP (1–3),

  ISBN: 0-13-018380-6
- Stevens, Richard: TCP/IP Illustrated {I | II | III}
- Tanenbaum, Andrew: Computer Networks,

  ISBN: 0-13-066102-3
- perldoc
- Beispielprogramme der Perl–Pakete

So long and thanks for all the fish

Fragen? Anmerkungen? Kritik?

$\Rightarrow$ Jetzt oder später an {mm|tho}@koeln.ccc.de