Hochschule Darmstadt
Fachbereich Informatik

Systemprogrammierung mit Perl

12. Netzwerk Programmierung

Dieses Kapitel stammt aus dem Buch:
"Programming the Network with Perl" von Paul Barry, Wiley Verlag

Hochschule Darmstadt
Fachbereich Informatik

Systemprogrammierung mit Perl

12.1 Die Pcap Library

Dieses Kapitel stammt aus dem Buch:
"Programming the Network with Perl" von Paul Barry, Wiley Verlag

Pcap und NetPacket

- Zur Untersuchung von Netzwerken gibt es zwei wichtige Bibliotheken
- Die Pcap Library wird meist bei Unix oder Linux Systemen mitgeliefert, für Windows ist sie als OpenSource Paket und www.winpcap.org zu finden
- Die Modul Net::Pcap und Net::PcapUtils stellen Schnittstellen in Perl zu der in C geschriebenen Pcap Biblithek zur Verfügung
- Die Pcap Bibliothek gestattet es Pakete vom Ethernet zu lesen, das Netpacket Modul hilft die gelesenen Module zu analysieren

© Gerhard Raffius, WS 2006-07, h_da - Fachbereich Informatik

Ein erstes Beispiel

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

 use strict;
 use Net::PcapUtils;
 use NetPacket::Ethernet;

 sub process_pkt {
     my($arg, $hdr, $pkt) = @_;

     my $eth_obj = NetPacket::Ethernet->decode($pkt);
     print("$eth_obj->{src_mac} : $eth_obj->{dest_mac} $eth_obj->{type}\n");
 }

 Net::PcapUtils::loop( \&process_pkt,
      DEV => '\Device\NPF_{6E8A8FC0-DC3C-403A-86AB-FF20A2963E0C}');
```

- die loop Methode erhält als Parameter eine Callback Methode, die wiederum als eines Ihrer Argumente die Rohdaten enthält

Ermittlung des Devices

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

use strict;
use Net::Pcap;
my ($error, %description);
foreach(Net::Pcap::findalldevs(\$error, \%description)) {
      print "$_\n   $description{$_}\n\n";
}
print $error if defined $error;
```

- Mit der Methode findalldevs kann man einen Überblick über die Devices und Ihre Namen im System erhalten
- Dies ist besonders für Windows wichtig, da dort der Devicename des Ethernetdevices nicht 'eth0' ist

© Gerhard Raffius, WS 2006-07, h_da - Fachbereich Informatik

Default Parameter für Loop und Open

```
my %args = (
	SNAPLEN => 100,		# Num bytes to capture from packet
	PROMISC => 1,		# Operate in promiscuous mode?
	TIMEOUT => 1000,	# Read timeout (ms)
	NUMPACKETS => -1,	# Pkts to read (-1 = loop forever)
	FILTER => '',		# Filter string
	USERDATA => '',		# Passed as first arg to callback fn
	SAVEFILE => '',		# Default save file
	DEV => '',		# Network interface to open
);
```

- Die Default Parameter können beim Aufruf von open oder loop überschrieben werden

© Gerhard Raffius, WS 2006-07, h_da - Fachbereich Informatik

Eine etwas verbesserte Callback Funktion

```
sub got_a_packet {
   my ( $user_arg, $header, $packet ) = @_;

   print "Got a packet!\n\n" ;
   print "The user argument is: ", $user_arg, "\n";
   print "The header data is: \n";
   foreach my $name (sort keys %{$header})
   {
      print "    $name -> ${$header}{$name}\n";
   }
 #   print "The packet data is:   ", $packet, "\n";
}
```

- Die wesentliche Information steckt im Datenpaket und kann mithilfe der NetPacket Module decodiert werden

© Gerhard Raffius, WS 2006-07, h_da - Fachbereich Informatik

Ein erster Ethernetsnooper: Die Pakettypen

```
our %type_totals = ();

our %type_desc = (
     0x0800 => 'IPv4',
     0x0806 => 'ARP',
     0x809B => 'AppleTalk',
     0x814C => 'SNMP',
     0x86DD => 'IPv6',
     0x880B => 'PPP',
     0x8137 => 'NOVELL1',
     0x8138 => 'NOVELL2',
     0x8035 => 'RARP',
     0x876B => 'TCP/IPc'
);
```


© Gerhard Raffius, WS 2006-07, h_da - Fachbereich Informatik

Die Callback Funktion zum Zählen der Pakettypen

```
sub got_a_packet {
   my ( $user_arg, $header, $packet ) = @_;

   my $frame = NetPacket::Ethernet->decode( $packet );

   if ($frame->{type} < 1501)    {
     $type_totals{ 1500 }++;
   }
   else    {
     $type_totals{ $frame->{type} }++;
   }

   $num_packets++;
}
```

- Alle Pakete mit einem Frametyp < 1500 werden pauschal gezählt

© Gerhard Raffius, WS 2006-07, h_da - Fachbereich Informatik

Die Display Funktion

```
sub display_results {
   print "$num_packets frames processed.\n\n";

   foreach my $etype ( sort keys %type_desc )
   {
     print "$type_desc{$etype} generated ";
     if ( exists $type_totals{$etype} )          {
        print "$type_totals{$etype} packets.\n"
     }
     else {
        print "no packets.\n";
     }
   }
      print "\nNon Ethernet II (DIX) frames generated";
   print " $type_totals{1500} packets.\n";
}
```

- Es wird eine sortierte Liste der Pakettypen ausgegeben

© Gerhard Raffius, WS 2006-07, h_da - Fachbereich Informatik

Das Hauptprogramm mit dem Loop Aufruf

```
$type_totals{ 1500 } = 0;

my $status = Net::PcapUtils::loop(
            \&got_a_packet,
            NUMPACKETS => 100,
            DEV => '\Device\NPF_{6E8A8FC0-DC3C-403A-86AB-FF20A2963E0C}'
          );
if ( $status ) {
   print "Net::PcapUtils::loop returned: $status\n";
}
else {
   display_results;
}
```

- In diesem Fall kehrt Loop zurück, wenn die Anzahl der Pakete erreicht ist

© Gerhard Raffius, WS 2006-07, h_da - Fachbereich Informatik

Das Hauptprogramm mit einer Zeitüberwachung: Nutzung von open und next

```
my $pkt_descriptor = Net::PcapUtils::open(
              DEV => '\Device\NPF_{6E8A8FC0-DC3C-403A-86AB-FF20A2963E0C}');

if ( !ref( $pkt_descriptor ) )
{
    print "Net::PcapUtils::open returned: $pkt_descriptor\n";
    exit;
}
```

- Open liefert eine gültige Referenz auf einen Packet Descriptor oder einen Fehlertext zurück

Pakete lesen mit next

```
my $minute = 1;
my $now = time;
my $then = $now + (60 * $minute);
my ( $next_packet, %next_header );
$type_totals{ 1500 } = 0;

while ( ($now = time) < $then )
{
   ( $next_packet, %next_header ) =
   Net::PcapUtils::next( $pkt_descriptor );
   got_a_packet( $next_packet );
}

display_results;
```

- next kehrt nach jedem empfangenen Paket zurück, so dass eine Zeitüberwachung möglich ist

© Gerhard Raffius, WS 2006-07, h_da - Fachbereich Informatik

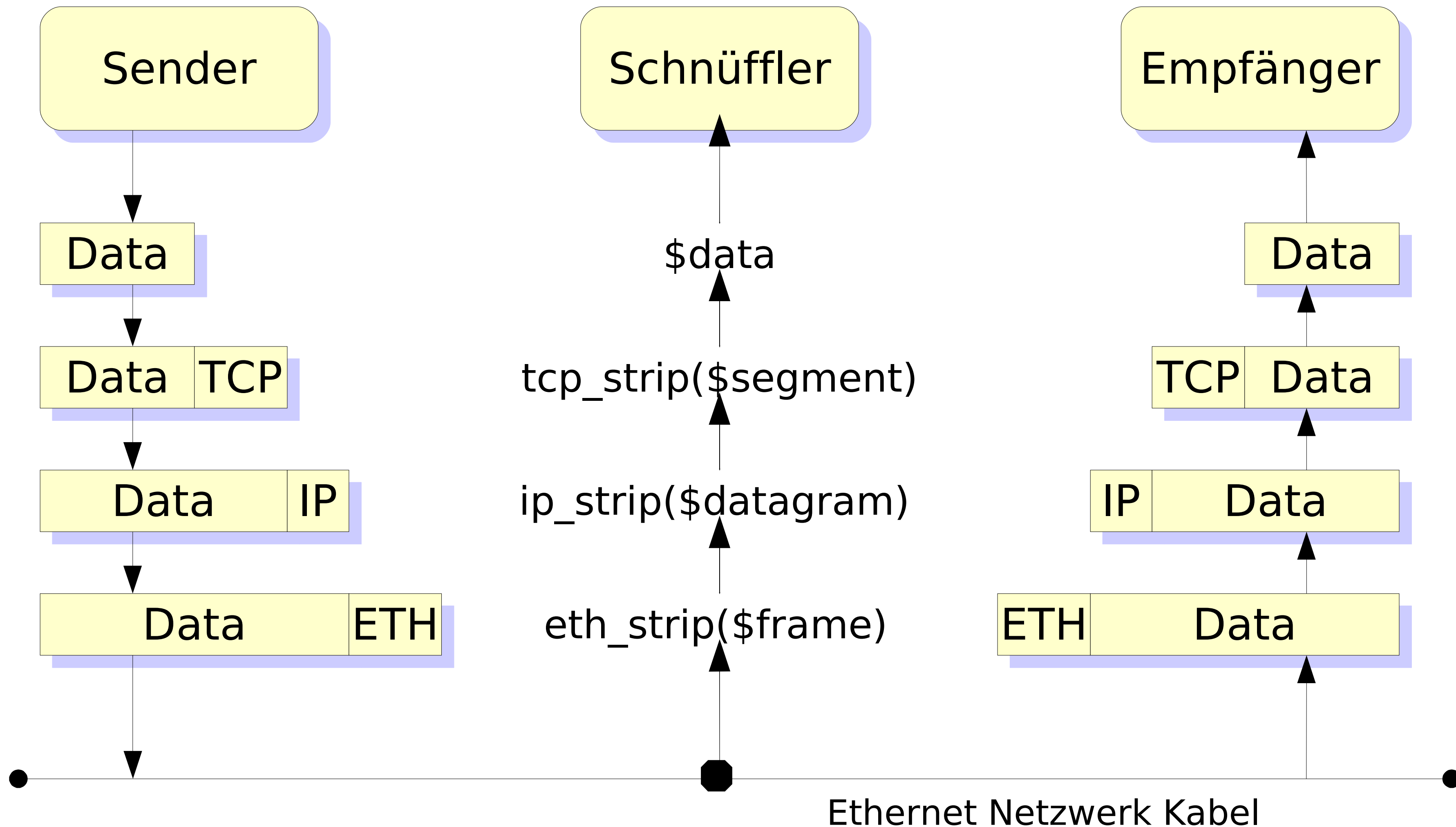
Hochschule Darmstadt
Fachbereich Informatik

Systemprogrammierung mit Perl

12.2 Ethernet Snooping

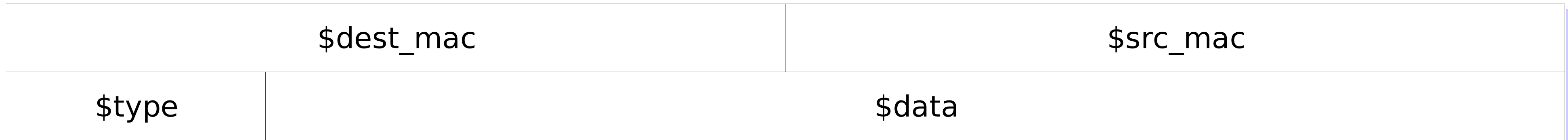
Dieses Kapitel stammt aus dem Buch:
"Programming the Network with Perl" von Paul Barry, Wiley Verlag

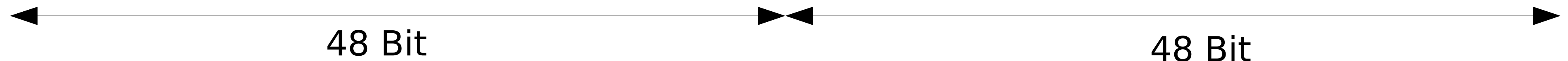
Wie funktioniert Snooping



© Gerhard Raffius, WS 2006-07, h_da - Fachbereich Informatik

Aufbau des Ethernet Frames





ETH_TYPE_IP	Internet Protokoll Version 4 Paket
ETH_TYPE_ARP	Adress Resolution Protokoll
ETH_TYPE_APPLETALK	Apple Computer AppleTalk Paket
ETH_TYPE_SNMP	Simple Network Management Protokoll Paket
ETH_TYPE_IPv6	Internet Protokoll Version 6 Paket
ETH_TYPE_PPP	Point To Point Prtokoll Paket

© Gerhard Raffius, WS 2006-07, h_da - Fachbereich Informatik

Die MAC Adressen der Teilnehmer analysieren

```
our $num_packets = 0;

our %src_hosts = ();
our %dest_hosts = ();

sub got_a_packet {
   my $packet = shift;

   my $frame = NetPacket::Ethernet->decode( $packet );

   $src_hosts{ $frame->{src_mac} }++;
   $dest_hosts{ $frame->{dest_mac} }++;

   $num_packets++;
}
```

- Perl Hashes sind ein sehr nützliches Hilfsmittel in der Datenanalyse

Die zughörige Display Routine

```
sub display_results {
    print "$num_packets frames processed.\n\n";

    print "The host statistics are:\n\nSources:\n\n";

    foreach my $host (sort keys %src_hosts )
    {
        print "Host: $host, Count: $src_hosts{$host}.\n";
    }

    print "\nDestinations:\n\n";

    foreach my $host (sort keys %dest_hosts )
    {
        print "Host: $host, Count: $dest_hosts{$host}.\n";
    }
}
```


© Gerhard Raffius, WS 2006-07, h_da - Fachbereich Informatik

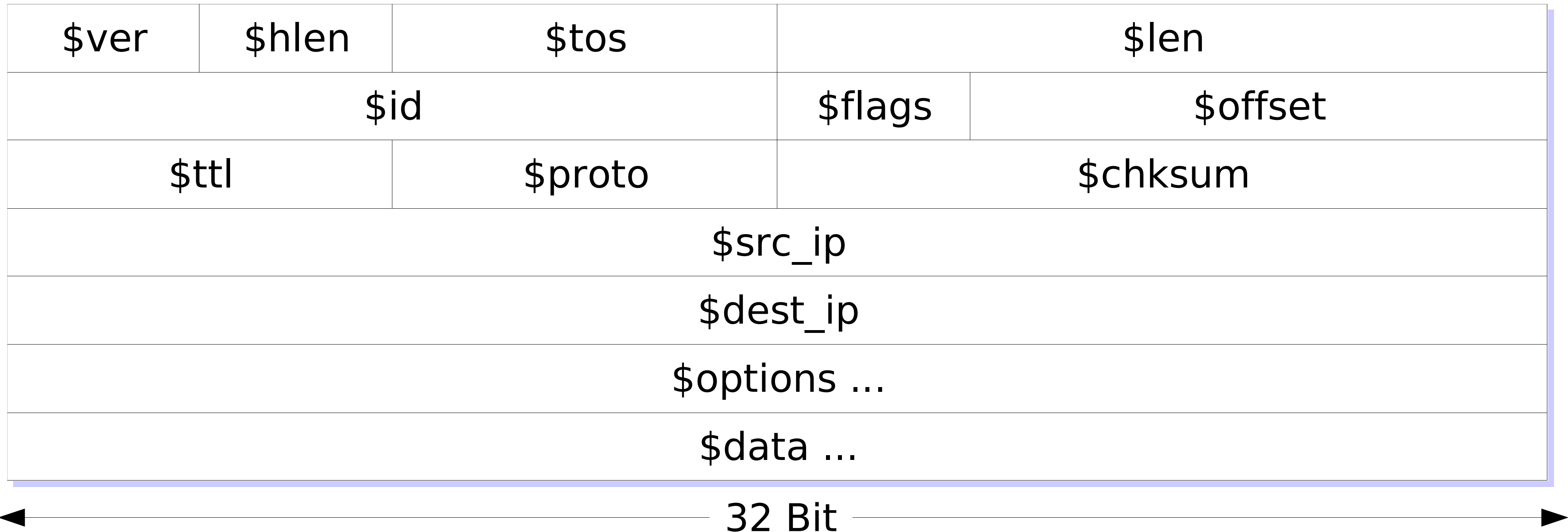
Hochschule Darmstadt
Fachbereich Informatik

Systemprogrammierung mit Perl

12.3 IP Snooping

Dieses Kapitel stammt aus dem Buch:
"Programming the Network with Perl" von Paul Barry, Wiley Verlag

Das NetPacket::IP Datagram Format



Die Definition der Felder im IP Datagramm

ver	The IP version number of this packet.
hlen	The IP header length of this packet.
flags	The IP header flags for this packet.
foffset	The IP fragment offset for this packet.
tos	The type-of-service for this IP packet.
len	The length (including length of header) in bytes for this packet.
id	The identification (sequence) number for this IP packet.
ttl	he time-to-live value for this packet.
proto	The IP protocol number for this packet.
cksum	The IP checksum value for this packet.
src_ip	The source IP address for this packet in dotted-quad notation.
dest_ip	The destination IP address for this packet in dotted-quad notation.
options	Any IP options for this packet.
data	The encapsulated data (payload) for this IP packet.

© Gerhard Raffius, WS 2006-07, h_da - Fachbereich Informatik

Interessanter als die Mac Adressen sind manchmal die IP Adressen

```
sub got_a_packet {
   my $packet = shift;
     my $frame = NetPacket::Ethernet->decode( $packet );

   $src_hosts{ $frame->{src_mac} }++;
   $dest_hosts{ $frame->{dest_mac} }++;
   if ( $frame->{type} == NetPacket::Ethernet::ETH_TYPE_IP )
   {
      my $ip_datagram = NetPacket::IP->decode(
            NetPacket::Ethernet::eth_strip( $packet ) );

      $e2ip{ $frame->{src_mac} } = $ip_datagram->{src_ip};
      $e2ip{ $frame->{dest_mac} } = $ip_datagram->{dest_ip};
   }
    $num_packets++;
}
my $pkt_descriptor = Net::PcapUtils::open(
            FILTER => 'ip',
            DEV => '\Device\NPF_{6E8A8FC0-DC3C-403A-86AB-FF20A2963E0C}' );
```

- Wir nutzen jetzt die Filtereigenschaften des Pcap Pakets und sammeln nur noch IP Datagramme

© Gerhard Raffius, WS 2006-07, h_da - Fachbereich Informatik

IP Datagramme analysieren und das TimeToLive Feld auswerten

```
our %ttl_totals = ();
our $num_datagrams = 0;

sub got_a_packet {
   my $packet = shift;

   my $ip_datagram = NetPacket::IP->decode(
         NetPacket::Ethernet::eth_strip( $packet ) );

   $ttl_totals{ $ip_datagram->{ttl} }++;

   $num_datagrams++;
}
```

- Das TimeToLive Feld wird von jedem Router um eins dekrementiert und bei einem Wert von 0 wird das Datagramm vernichtet.
- Der Startwert sollte 60 sein

© Gerhard Raffius, WS 2006-07, h_da - Fachbereich Informatik

TimeToLive Daten auswerten 1

```
sub display_results {
    print "$num_datagrams datagrams processed.\n\n";

    my $min_ttl = 256;
    my $max_ttl = 0;
    my $average_ttl = 0;

    while ( my ( $ttl_key, $ttl_value ) = each %ttl_totals )
    {
        if ( $ttl_key < $min_ttl )  {
            $min_ttl = $ttl_key;
        }
        if ( $ttl_key > $max_ttl )  {
            $max_ttl = $ttl_key;
        }
        $average_ttl =  $average_ttl + ( $ttl_key * $ttl_value );
    }
    $average_ttl = ( $average_ttl / $num_datagrams );
```

TimeToLive Daten auswerten 2

```
    print "Minimum TTL value: $min_ttl\n";
    print "Maximum TTL value: $max_ttl\n";
    print "Average TTL value: $average_ttl\n\n";
    print "TTL distribution analysis:\n\n";

    foreach my $ttlkey ( sort {$a <=> $b} keys %ttl_totals )
    {
        print "TTL: $ttlkey, ";
        print "frequency: $ttl_totals{$ttlkey}.\n";
    }
}
```


© Gerhard Raffius, WS 2006-07, h_da - Fachbereich Informatik

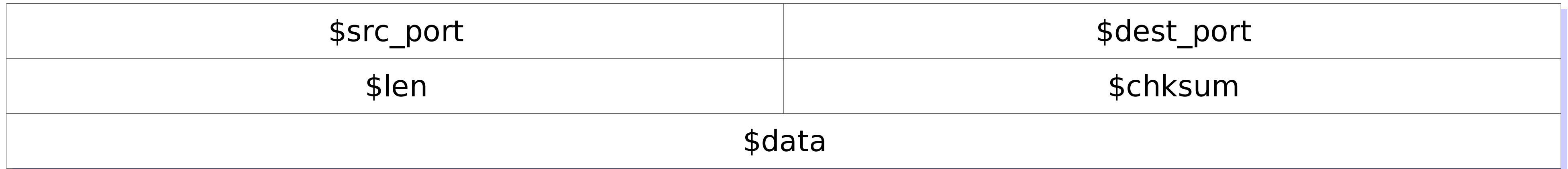
Hochschule Darmstadt
Fachbereich Informatik

Systemprogrammierung mit Perl

12.4 UDP Snooping

Dieses Kapitel stammt aus dem Buch:
"Programming the Network with Perl" von Paul Barry, Wiley Verlag

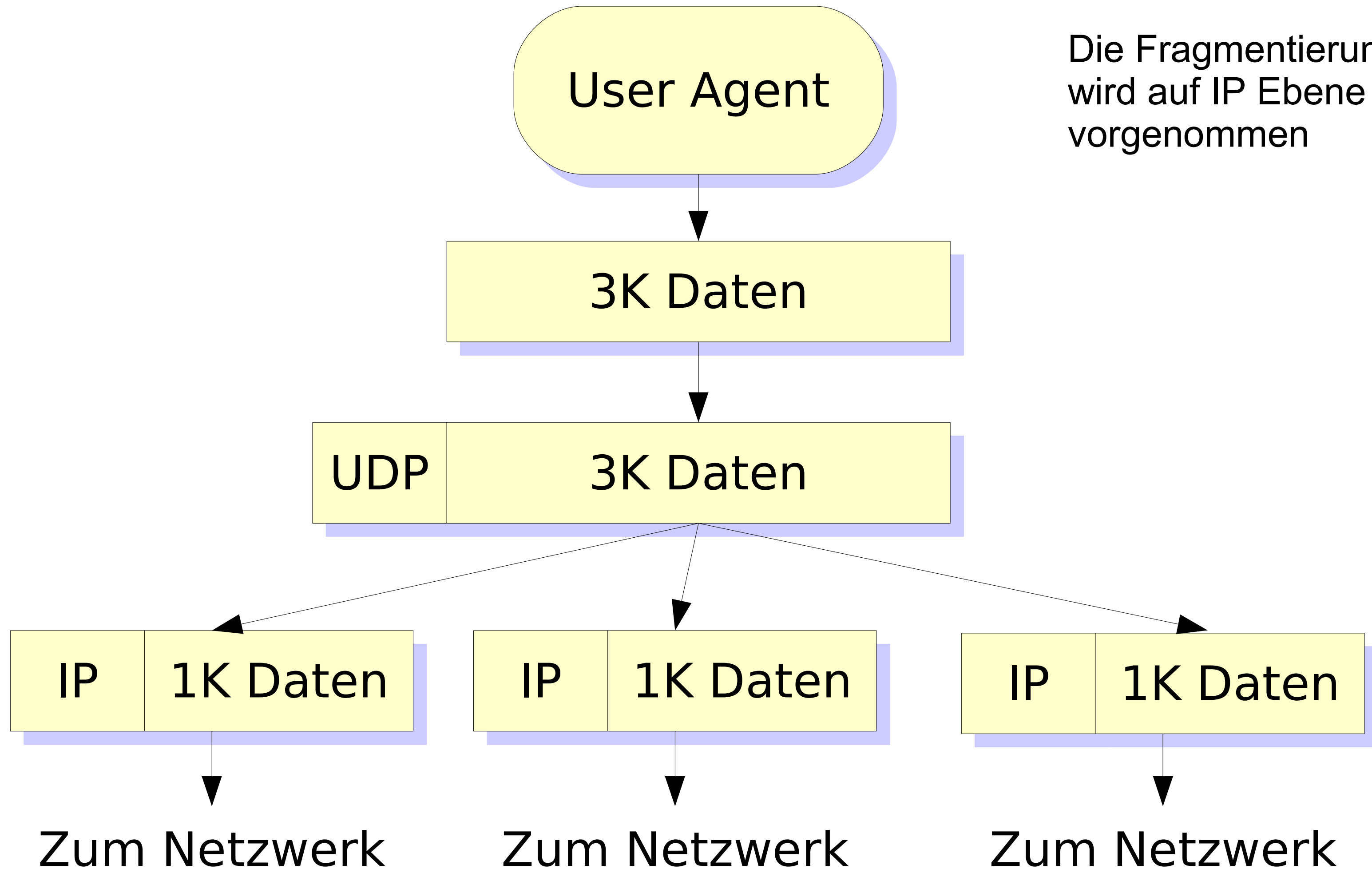
Das Netpacket::UDP Datagramm Format



32 Bit

© Gerhard Raffius, WS 2006-07, h_da - Fachbereich Informatik

UDP/IP Fragmentierung



Die Fragmentierung wird auf IP Ebene vorgenommen

© Gerhard Raffius, WS 2006-07, h_da - Fachbereich Informatik

Fragmentierung suchen

```
sub got_a_packet {
   my $packet = shift;
   my $ip_datagram = NetPacket::IP->decode(
         NetPacket::Ethernet::eth_strip( $packet ) );
   $num_datagrams++;
   if( $ip->datagram->{foffset} > 0 ) {
          $num_fragments++;
          exit;
   }
   $ip_type_totals{ $ip_datagram->{proto} }++;
}
```

- Das Feld flags im IP Datagramm zeigt an, ob ein Datagramm Teil einer Fragmentierung ist
  - ⇨ 000 bedeutet, das Datagramm ist möglicherweise fragmentiert und dies ist das einzige oder letzte Fragment
  - ⇨ 001 bedeutet, das Datagramm ist fragmentiert und weitere Fragmente folgen
  - ⇨ 010 (dezimal 2) bedeutet, das Datagramm darf nicht fragmentiert werden

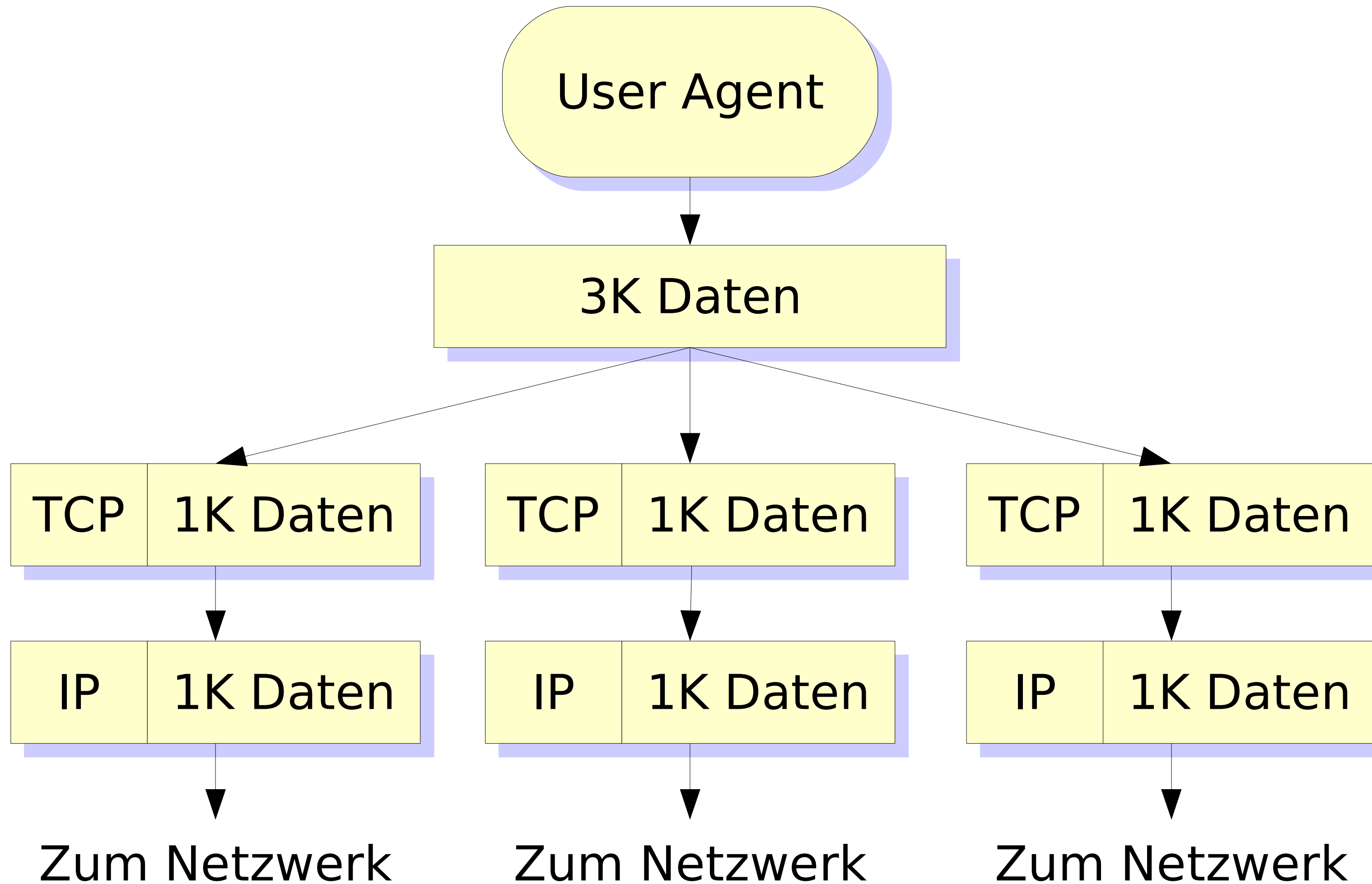
Hochschule Darmstadt
Fachbereich Informatik

Systemprogrammierung mit Perl

12.5 TCP Snooping

Dieses Kapitel stammt aus dem Buch:
"Programming the Network with Perl" von Paul Barry, Wiley Verlag

TCP/IP Fragmentierung



© Gerhard Raffius, WS 2006-07, h_da - Fachbereich Informatik

Die Definition der Felder im TCP Datagramm

src_port	The source TCP port for the packet.
dest_port	The destination TCP port for the packet.
seqnum	The TCP sequence number for this packet.
acknum	The TCP acknowledgement number for this packet.
hlen	The header length for this packet.
reserved	The 6-bit ``reserved'' space in the TCP header.
flags	Contains the urg, ack, psh, rst, syn, fin, ece and cwr flags for this packet.
winsize	The TCP window size for this packet.
cksum	The TCP checksum.
urg	The TCP urgent pointer.
options	Any TCP options for this packet in binary form.
data	The encapsulated data (payload) for this packet.

Einige wichtige TCP Portnummern

rje	5/tcp	Remote Job Entry
echo	7/tcp	Echo
discard	9/tcp	Discard
systat	11/tcp	Active Users
daytime	13/tcp	Daytime (RFC 867)
qotd	17/tcp	Quote of the Day
msp	18/tcp	Message Send Protocol
ftp-data	20/tcp	File Transfer [Default Data]
ftp	21/tcp	File Transfer [Control]
ssh	22/tcp	SSH Remote Login Protocol
telnet	23/tcp	Telnet
	24/tcp	any private mail system
smtp	25/tcp	Simple Mail Transfer
msg-auth	31/tcp	MSG Authentication
dsp	33/tcp	Display Support Protocol
	35/tcp	any private printer server
time	37/tcp	Time
inger	79/tcp	Finger
http	80/tcp	World Wide Web HTTP
hosts2-ns	81/tcp	HOSTS2 Name Server

TCP Pakete analysieren

```
sub got_a_packet {
   my $packet = shift;

   my $tcp_segment = NetPacket::TCP->decode(
                NetPacket::IP::ip_strip(
                     NetPacket::Ethernet::eth_strip( $packet ) ) );

   if ($tcp_segment->{src_port} < 1024)    {
      $tcp_proto_totals{ $tcp_segment->{src_port} }++;
   }

   if ($tcp_segment->{dest_port} < 1024)     {
      $tcp_proto_totals{ $tcp_segment->{dest_port} }++;
   }

   $num_datagrams++;
}
```


© Gerhard Raffius, WS 2006-07, h_da - Fachbereich Informatik

Filterung beim Messageempfang

```
my $pkt_descriptor = Net::PcapUtils::open(
            FILTER => 'tcp',
            SNAPLEN => 120,
            DEV => '\Device\NPF_{6E8A8FC0-DC3C-403A-86AB-FF20A2963E0C}' );
```

- Wir wollen nun nur TCP Pakete empfangen und stellen daher den Filter im Open auf 'tcp'

Analyse und Ausgabe der Daten

```
sub display_results {
    print "$num_datagrams segments processed.\n\n";
    my %tcp_desc = ();

    open WELLKNOWN_TCP, 'well-known-tcp' or die "Ooops: $!";

    while ($_ = <WELLKNOWN_TCP>)    {
        chomp;
        m[(\d+)/tcp];
        $tcp_desc{$1} = $_;
    }
    close WELLKNOWN_TCP;

    foreach my $port (sort keys %tcp_proto_totals)    {
        print "Port: $port, ";
        print "generated $tcp_proto_totals{$port} segments.\n";
        print "Protocol-port: $tcp_desc{$port}.";
        print "\n\n";
    }
}
```

Signale in Perl

- In Perl kann ein Signalhandler für ein Signal installiert werden, indem man die Referenz auf den Handler in den Hash %SIG schreibt
- Mit der Funktion "kill Signal, PidList" kann man einer Liste von Prozessen ein Signal schicken
- Die Funktion "alarm SEC" löst das ALRM Signal nach SEC Sekunden aus

0	ZERO
1	HUP
2	INT
3	
4	ILL
5	NUM05
6	NUM06
7	NUM07
8	FPE
9	KILL
10	NUM10
11	SEGV
12	NUM12
13	PIPE
14	ALRM
15	TERM
16	NUM16
17	NUM17
18	NUM18
19	NUM19
20	CHLD
21	QUIT
22	ABRT
23	STOP
24	NUM24
25	CONT

Verbesserung der Zeitüberwachung

```
$SIG{ALRM} = sub { die; };

 eval
 {
    alarm( ( 60 * $minutes ) + 1 );

    my $now = time;
    my $then = $now + (60 * $minutes);

    while ( ($now = time) < $then )
    {
       ( $next_packet, %next_header ) =
             Net::PcapUtils::next( $pkt_descriptor );
       got_a_packet( $next_packet );
    }
    alarm(0);
 };
 alarm(0);
```


© Gerhard Raffius, WS 2006-07, h_da - Fachbereich Informatik

Hochschule Darmstadt
Fachbereich Informatik

Systemprogrammierung mit Perl

13. Client Server Programmierung mit Sockets

Dieses Kapitel stammt aus dem Buch:
"Programming the Network with Perl" von Paul Barry, Wiley Verlag

Ein erster UDP Server (1)

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

# udp_s1

use strict;
use Socket;

use constant SIMPLE_UDP_PORT => 4001;
use constant MAX_RECV_LEN    => 1500;
use constant LOCAL_INETNAME  => 'localhost';

my $trans_serv = getprotobyname( 'udp' );

my $local_host = gethostbyname( LOCAL_INETNAME );
my $local_port = SIMPLE_UDP_PORT;
my $local_addr = sockaddr_in( $local_port, $local_host );

socket( UDP_SOCK, PF_INET, SOCK_DGRAM, $trans_serv );

bind( UDP_SOCK, $local_addr );
```


© Gerhard Raffius, WS 2006-07, h_da - Fachbereich Informatik

Hilfsvariablen und Funktionen

- SIMPLE_UDP_PORT
  - ⇨ Die Port Nummer über die wir kommunizieren wollen
- MAX_RECV_LEN
  - ⇨ Die maximale Anzahl Zeichen die wir erhalten wollen wollen
- LOCAL_INETNAME
  - ⇨ Der Name unseres Servers
- gethostbyname
  - ⇨ Diese Funktion liefert uns die INET Adresse wenn wir Ihr einen Namen übergeben
- sockaddr_in
  - ⇨ Diese Funktion erzeugt uns aus der INET Adresse und der Portnummer eine vollständige wenn sie im skalaren Kontext aufgerufen wird
  - ⇨ Im Listenkontext erzeugt die Funktion INET Adresse und Portnummer aus der TCP/IP Adresse
  - ⇨ die Funktionen pack_sockadr_in und unpack_sockadr_in sind lesbarer

Die Funktion socket

- Die Funktion socket erzeugt uns ein Socket Handle, über das wir kommunizieren können
  - ⇨ socket SOCKET,DOMAIN,TYPE,PROTOCOL
- SOCKET
  - ⇨ der Name des Socket Handles. Es ist Konvention für den Namen nur Großbuchstaben zu benutzen
- DOMAIN
  - ⇨ Die Protokollfamilie die benutzt wird. In unserem Fall können wir die in socket.pm definierte Konstante PF_INET für die INET Protokollfamilie nutzen
- TYPE
  - ⇨ Als Typ stehen zur Verfügung SOCK_STREAM und SOCK_DGRAM
- PROTOCOL
  - ⇨ die interne Transport Service Protokollnummer. Wir erhalten sie durch einen Aufruf von getprotobyname('udp')

© Gerhard Raffius, WS 2006-07, h_da - Fachbereich Informatik

Die Funktionen bind, recv und send

- Die Funktion bind verbindet eine Netzwerkadresse mit einem Socket
  - ⇨ bind SOCKET,NAME
- Die Funktion recv empfängt Daten von einem Sockethandle und schreibt diese in die skalare Variable bis zur maximalen Länge. Die Flags entsprechen den jeweiligen Systemflags
  - ⇨ recv SOCKET,SCALAR,LENGTH,FLAGS
- Die Funktion send verschickt Daten über das Sockethandle an die Adresse TO
  - ⇨ send SOCKET,MSG,FLAGS,TO
  - ⇨ Bei verbindungsorientierten Protokollen kann der Parameter TO entfallen

© Gerhard Raffius, WS 2006-07, h_da - Fachbereich Informatik

Ein erster UDP Server (2)

```
my $data;

while( 1 )
{
   my $from_who = recv( UDP_SOCK, $data, MAX_RECV_LEN, 0 );

   if ( $from_who )  {
      my ( $the_port, $the_ip ) = sockaddr_in( $from_who );

      warn 'Received from ', inet_ntoa( $the_ip ), ": $data\n";
   }
   else  {
      warn "Problem with recv: $!\n";
   }
}
```

- inet_ntoa übersetzt uns die binäre Form der Netzwerkadresse in die übliche Textform

© Gerhard Raffius, WS 2006-07, h_da - Fachbereich Informatik

Ein erster UDP Client

```
use strict;
use Socket;

use constant SIMPLE_UDP_PORT => 4001;
use constant REMOTE_HOST     => 'localhost';

my $trans_serv  = getprotobyname( 'udp' );
my $remote_host = gethostbyname( REMOTE_HOST );
my $remote_port = SIMPLE_UDP_PORT;
my $destination = sockaddr_in( $remote_port, $remote_host );

socket( UDP_SOCK, PF_INET, SOCK_DGRAM, $trans_serv );

my $data = "This is a simple UDP message";

send( UDP_SOCK, $data, 0, $destination );

close UDP_SOCK;
```


© Gerhard Raffius, WS 2006-07, h_da - Fachbereich Informatik

Ein verbesserter Server (1)

```
use constant SIMPLE_UDP_PORT => 4001;
use constant MAX_RECV_LEN    => 1500;

my $local_port = shift || SIMPLE_UDP_PORT;
my $trans_serv = getprotobyname( 'udp' );
my $local_addr = sockaddr_in( $local_port, INADDR_ANY );

socket( UDP_SOCK, PF_INET, SOCK_DGRAM, $trans_serv )
    or die "udp_s2: socket creation failed: $!\n";

bind( UDP_SOCK, $local_addr )
    or die "udp_s2: bind to address failed: $!\n";
```

- Die Socket Library ist so geschrieben, dass bei erfolgreicherAusführung der Wert True zurückgeben wird. Die Fehlernummer steht jeweils in $!
- Die Portnummer kann nun auch als Kommandozeilen Parameter übergeben werden

© Gerhard Raffius, WS 2006-07, h_da - Fachbereich Informatik

Ein verbesserter Server (2)

```
my $data;

warn "Server starting up on port: $local_port.\n";

while( 1 ){
   my $from_who = recv( UDP_SOCK, $data, MAX_RECV_LEN, 0 );

   if ( $from_who )    {
      my ( $the_port, $the_ip ) = sockaddr_in( $from_who );
      my $remote_name = gethostbyaddr( $the_ip, AF_INET ) || inet_ntoa( $the_ip );

      warn "Received from $remote_name: $data\n";
   }
   else  {warn "Problem with recv: $!\n";    }
}
```

- gethostbyaddr liefert uns entweder den Namen des Rechners aus der IP Adresse oder den Wert FALSE

© Gerhard Raffius, WS 2006-07, h_da - Fachbereich Informatik

Ein verbesserter Client

```
use constant SIMPLE_UDP_PORT => 4001;
use constant REMOTE_HOST     => 'localhost';

my $remote = shift || REMOTE_HOST;
my $remote_port = shift || SIMPLE_UDP_PORT;
my $trans_serv  = getprotobyname( 'udp' );

my $remote_host = gethostbyname( $remote )  or die "udp_c2: name lookup failed: $remote\n";

my $destination = sockaddr_in( $remote_port, $remote_host );

socket( UDP_SOCK, PF_INET, SOCK_DGRAM, $trans_serv )
       or die "udp_c2: socket creation failed: $!\n";

my $data = "This is a simple UDP message";

send( UDP_SOCK, $data, 0, $destination ) or warn "udp_c2: send to socket failed.\n";

close UDP_SOCK or die "udp_c2: close socket failed: $!\n";
```

Empfangen und Senden beim Server

```
my $data;

warn "Server starting up on port: $local_port.\n";

while( 1 ) {
   my $from_who = recv( UDP_SOCK, $data, MAX_RECV_LEN, 0 );

   if ( $from_who )    {
      my ( $the_port, $the_ip ) = sockaddr_in( $from_who );
      my $remote_name = gethostbyaddr( $the_ip, AF_INET ) || inet_ntoa( $the_ip );
      warn "Received from $remote_name: ", length( $data ),  ' -> ', substr( $data, 0, 39 ), "\n";

      sleep(3);

      warn "Sending back to client ... \n";
      send( UDP_SOCK, $data, 0, $from_who ) or warn "udp_s5: send to socket failed.\n";
   }
   else  {  warn "Problem with recv: $!\n";  }
}
```


© Gerhard Raffius, WS 2006-07, h_da - Fachbereich Informatik

Senden und empfangen beim Client

```
my $msg_count = 1;
my $big_chunk = 'x' x 65000;

while ( $msg_count < 11 ) {
    my $data = $msg_count . ' -> ' . $big_chunk;

    warn "Sending $msg_count to server ...\n";
    send( UDP_SOCK, $data, 0, $destination ) or warn "udp_c5: send to socket failed: $msg_count\n";
    sleep(1);

    my $from_who = recv( UDP_SOCK, $data, MAX_RECV_LEN, 0 );

    if ( $from_who )    {
        my ( $the_port, $the_ip ) = sockaddr_in( $from_who );
        my $remote_name = gethostbyaddr( $the_ip, AF_INET ) || inet_ntoa( $the_ip );
        warn "Received from $remote_name: ", length( $data ), ' -> ', substr( $data, 0, 39 ), "\n";
    }
    else   {  warn "Problem with recv: $!\n";  }
    $msg_count++;
}
close UDP_SOCK  or die "udp_c5: close socket failed: $!\n";
```

Receive Timeout beim Client

```
send( UDP_SOCK, $data, 0, $destination ) or warn "udp_c6: send to socket failed: $msg_count\n";
sleep(1);

$SIG{ALRM} = sub { die "recv timeout\n"; };
alarm( 5 );

eval {
    my $from_who = recv( UDP_SOCK, $data, MAX_RECV_LEN, 0 );

    if ( $from_who ) {
        my ( $the_port, $the_ip ) = sockaddr_in( $from_who );
        my $remote_name = gethostbyaddr( $the_ip, AF_INET ) || inet_ntoa( $the_ip );
        warn "Received from $remote_name: ", length( $data ),    ' -> ', substr( $data, 0, 39 ), "\n";
    }
    else  {  warn "Problem with recv: $!\n";  }
    alarm( 0 );
};

if ($@)    {
    die "udp_c6: $@\n" unless $@ =~ /recv timeout/;
    warn "udp_c6: recv timed out, cancelling ...\n";
}
```


© Gerhard Raffius, WS 2006-07, h_da - Fachbereich Informatik

Ein Objektorientierter Server

```
use strict;
use IO::Socket;
use constant $MAXLEN => 1024;
use constant $PORTNO => 5151;

my($sock, $oldmsg, $newmsg, $hisaddr, $hishost);

$sock = IO::Socket::INET->new(LocalPort => $PORTNO, Proto => 'udp')
    or die "socket: $@";

print "Awaiting UDP messages on port $PORTNO\n";
$oldmsg = "This is the starting message.";

while ($sock->recv($newmsg, $MAXLEN)) {
    my($port, $ipaddr) = sockaddr_in($sock->peername);
    $hishost = gethostbyaddr($ipaddr, AF_INET);
    print "Client $hishost said ``$newmsg''\n";
    $sock->send($oldmsg);
    $oldmsg = "[$hishost] $newmsg";
}
die "recv: $!";
```


© Gerhard Raffius, WS 2006-07, h_da - Fachbereich Informatik

Ein Objektorientierter UDP Client

```
use IO::Socket;
use constant $MAXLEN  => 1024;
use constant $PORTNO  => 5151;
use constant $TIMEOUT => 5;

my($sock, $server_host, $msg, $port, $ipaddr, $hishost);
$server_host = 'localhost';
$msg         = shift;
$sock = IO::Socket::INET->new(Proto => 'udp', PeerPort => $PORTNO, PeerAddr  => $server_host)
    or die "Creating socket: $!\n";
$sock->send($msg) or die "send: $!";

eval {
    local $SIG{ALRM} = sub { die "alarm time out" };
    alarm $TIMEOUT;
    $sock->recv($msg, $MAXLEN)      or die "recv: $!";
    alarm 0;
    1;  # return value from eval on normalcy
} or die "recv from $server_host timed out after $TIMEOUT seconds.\n";

($port, $ipaddr) = sockaddr_in($sock->peername);
$hishost = gethostbyaddr($ipaddr, AF_INET);
print "Server $hishost responded ``$msg''\n";
```


© Gerhard Raffius, WS 2006-07, h_da - Fachbereich Informatik

Hochschule Darmstadt
Fachbereich Informatik

Systemprogrammierung mit Perl

13.2 TCP Connections

Dieses Kapitel stammt aus dem Buch:
"Programming the Network with Perl" von Paul Barry, Wiley Verlag

Ein erster TCP Server (1)

```
use strict;
use Socket;

use constant SIMPLE_TCP_PORT => 4001;
use constant MAX_RECV_LEN    => 65536;

my $local_port = shift || SIMPLE_TCP_PORT;

my $trans_serv = getprotobyname( 'tcp' );

my $local_addr = sockaddr_in( $local_port, INADDR_ANY );

socket( TCP_SOCK, PF_INET, SOCK_STREAM, $trans_serv )
   or die "tcp_s1b: socket creation failed: $!\n";

setsockopt( TCP_SOCK, SOL_SOCKET, SO_REUSEADDR, 1 )
   or warn "tcp_s1b: could not set socket option: $!\n";
```


© Gerhard Raffius, WS 2006-07, h_da - Fachbereich Informatik

Ein erster TCP Server (2)

```
bind( TCP_SOCK, $local_addr )
    or die "tcp_s1b: bind to address failed: $!\n";

listen( TCP_SOCK, SOMAXCONN )
    or die "tcp_s1b: listen couldn't: $!\n";

warn "Server starting up on port: $local_port.\n";
```


© Gerhard Raffius, WS 2006-07, h_da - Fachbereich Informatik

Ein erster TCP Server (3)

```
while ( $from_who = accept( CLIENT_SOCK, TCP_SOCK ) )
{
   my ( $chunk, $data );

   recv( CLIENT_SOCK, $chunk, MAX_RECV_LEN, 0 );
   while ( $chunk )  {
      $data = $data . $chunk;
      recv( CLIENT_SOCK, $chunk, MAX_RECV_LEN, 0 );
   }

   my ( $the_port, $the_ip ) = sockaddr_in( $from_who );
   my $remote_name = gethostbyaddr( $the_ip, AF_INET ) || inet_ntoa( $the_ip );
   warn "Received from $remote_name: ", length( $data ), ' -> ', substr( $data, 0, 39 ), "\n";

   sleep( 3 );

   warn "Sending ", length( $data ), " back to client ... \n";
   send( CLIENT_SOCK, $data, 0 )  or warn "tcp_s1b: problem with send: $!\n";
}
continue {
   close CLIENT_SOCK   or warn "tcp_c1: close failed: $!\n";
}
close TCP_SOCK;
```


© Gerhard Raffius, WS 2006-07, h_da - Fachbereich Informatik

Ein erster TCP Client (1)

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

use strict;
use Socket;

use constant SIMPLE_TCP_PORT => 4001;
use constant REMOTE_HOST     => 'localhost';
use constant MAX_RECV_LEN    => 65536;

my $remote = shift || REMOTE_HOST;
my $remote_port = shift || SIMPLE_TCP_PORT;

my $trans_serv  = getprotobyname( 'tcp' );

my $remote_host = gethostbyname( $remote )
   or die "tcp_c1b: name lookup failed: $remote\n";

my $destination = sockaddr_in( $remote_port, $remote_host );
```


© Gerhard Raffius, WS 2006-07, h_da - Fachbereich Informatik

Ein erster TCP Client (2)

```
my $msg_count = 1;
my $big_chunk = 'x' x 65000;

while ( $msg_count < 11 )
{
    socket( TCP_SOCK, PF_INET, SOCK_STREAM, $trans_serv )
        or die "tcp_c1b: socket creation failed: $!\n";

    my $con_ok = connect( TCP_SOCK, $destination )
        or warn "tcp_c1b: connect to remote system failed: $!\n";

    next unless $con_ok;

    my $data = $msg_count . ' -> ' . $big_chunk;

    warn "Sending $msg_count ", length( $data ), " to server ...\n";

    send( TCP_SOCK, $data, 0 )  or warn "tcp_c1b: problem with send: $!\n";
    sleep( 1 );
    shutdown( TCP_SOCK, 1 );
```


© Gerhard Raffius, WS 2006-07, h_da - Fachbereich Informatik

Ein erster TCP Client (3)

```
  $data = '';
   my $chunk;

   recv( TCP_SOCK, $chunk, MAX_RECV_LEN, 0 );

   while ( $chunk )  {
      $data = $data . $chunk;
      recv( TCP_SOCK, $chunk, MAX_RECV_LEN, 0 );
   }

   my ( $the_port, $the_ip ) = sockaddr_in( $destination );
   my $remote_name = gethostbyaddr( $the_ip, AF_INET ) || inet_ntoa( $the_ip );
   warn "Received from $remote_name: ", length( $data ), ' -> ', substr( $data, 0, 39 ), "\n";

   close TCP_SOCK or warn "tcp_c1b: close failed: $!\n";
}
continue {
   $msg_count++;
}
```

Ein Server der Daten versendet

```
while ( accept( CLIENT_SOCK, TCP_SOCK ) )
{
    my $secs = 10;

    my $previous = select CLIENT_SOCK;
    $| = 1;
    select $previous;

    print CLIENT_SOCK "Sleeping for $secs seconds ... \n";
    sleep( $secs );
    print CLIENT_SOCK "I awake, only to die ... \n";

    close CLIENT_SOCK
        or warn "tcp_c1: close failed: $!\n";
}
close TCP_SOCK;
```

- Bei Datenströmen können die normalen Perlmechanismen zur Ein- und Ausgabe genutzt werden. Es ist jedoch sinnvoll für den Kanal autoflush einzustellen

© Gerhard Raffius, WS 2006-07, h_da - Fachbereich Informatik

Ein Client der Daten empfängt

```
use constant SIMPLE_TCP_PORT => 4001;
use constant REMOTE_HOST     => 'localhost';

my $remote = shift || REMOTE_HOST;
my $remote_port = shift || SIMPLE_TCP_PORT;
my $trans_serv  = getprotobyname( 'tcp' );
my $remote_host = gethostbyname( $remote )
   or die "tcp_c2: name lookup failed: $remote\n";
my $destination = sockaddr_in( $remote_port, $remote_host );

socket( TCP_SOCK, PF_INET, SOCK_STREAM, $trans_serv )
   or die "tcp_c2: socket creation failed: $!\n";
connect( TCP_SOCK, $destination )
   or die "tcp_c2: connect to remote system failed: $!\n";

while ( <TCP_SOCK> )
{
   print $_;
}
close TCP_SOCK;
```

Ein Objektorientierter Server

```
use strict;
use IO::Socket;
use constant SIMPLE_TCP_PORT => 4001;
my $port = shift || SIMPLE_TCP_PORT;

my $sock_obj = IO::Socket::INET->new( LocalPort   => $port,
                                      Proto       => 'tcp',
                                      Reuse       => 1,
                                      Listen      => SOMAXCONN )
    or die "oo_tcp_s3: could not create socket object: $!\n";
warn "OO Server starting up on port: ", $sock_obj->sockport, ".\n";

while ( my $client_obj = $sock_obj->accept ) {
    my $secs = 10;

    print $client_obj "Sleeping for $secs seconds ... \n";
    sleep( $secs );
    print $client_obj "I awake, only to die ... \n";

    $client_obj->close        or warn "oo_tcp_s3: close failed: $!\n";
}
$sock_obj->close;
```


© Gerhard Raffius, WS 2006-07, h_da - Fachbereich Informatik

Ein Objektorientierter Client

```
use strict;
use IO::Socket;

use constant SIMPLE_TCP_PORT => 4001;
use constant REMOTE_HOST     => 'localhost';

my $remote = shift || REMOTE_HOST;
my $remote_port = shift || SIMPLE_TCP_PORT;

my $sock_obj = IO::Socket::INET->new( PeerAddr => $remote,
                                      PeerPort => $remote_port,
                                      Proto    => 'tcp' )
    or die "oo_tcp_c3: could not create socket object: $!\n";

while ( <$sock_obj> )
{
    print $_;
}
$sock_obj->close;
```


© Gerhard Raffius, WS 2006-07, h_da - Fachbereich Informatik

Hochschule Darmstadt
Fachbereich Informatik

Systemprogrammierung mit Perl

14. Ein einfacher Webserver

Dieses Kapitel stammt aus dem Buch:
"Programming the Network with Perl" von Paul Barry, Wiley Verlag

Ein erster einfacher Webserver

```
use HTTP::Daemon;
use HTTP::Status;

my $d = HTTP::Daemon->new ( LocalPort => 8080) or die;

print "Please contact me at: <URL:", $d->url, ">\n";
while (my $c = $d->accept) {
     while (my $r = $c->get_request) {
        if ($r->method eq 'GET'  and $r->url->path eq "/") {

            # remember, this is *not* recommended practice :-)
            $c->send_file_response("c:/tmp/Test for Simple Server.html");
        }
        else {
            $c->send_error(RC_FORBIDDEN)
        }
     }
     $c->close;
     undef($c);
}
```


© Gerhard Raffius, WS 2006-07, h_da - Fachbereich Informatik