Read Linux networking code without C-ing

Kernel-Code abstrahieren und verstehen

Moritz Flüchter

05.07.2025

huhu Tübix :)

Warum eigentlich?

Warum eigentlich?

```c
int ping_recvmsg(struct sock *sk, struct msghdr *msg, size_t len, int flags,
         int *addr_len)
{
    struct inet_sock *isk = inet_sk(sk);
    int family = sk->sk_family;
    struct sk_buff *skb;
    int copied, err;

    pr_debug("ping_recvmsg(sk=%p,sk->num=%u)\n", isk, isk->inet_num);

    err = -EOPNOTSUPP;
    if (flags & MSG_OOB)
        goto out;

    if (flags & MSG_ERRQUEUE)
        return inet_recv_error(sk, msg, len, addr_len);

    skb = skb_recv_datagram(sk, flags, &err);
    if (!skb)
        goto out;
```

- Unbekannte Namen
- Viele Abkürzungen
- `*isk = inet_sk(sk);`
- Selten Kommentare
- Komische C Sachen

Warum eigentlich?

```
1   int ping_recvmsg(struct sock *sk, struct msghdr *msg, size_t len, int flags,
2           int *addr_len)
3   {
4       struct inet_sock *isk = inet_sk(sk);
5
6       pr_debug("ping_recvmsg(sk=%p,sk->num=%u)\n", isk, isk->inet_num);
7
8       err = -EOPNOTSUPP;
9       if (flags & MSG_OOB)
10          goto out;
11
12      if (flags & MSG_ERRQUEUE)
13          return inet_recv_error(sk, msg, len, addr_len);
14
15      skb = skb_recv_datagram(sk, flags, &err);
16      if (!skb)
17          goto out;
18
19      err = skb->len;
20      if (len < err)
```

Variablen deklarieren

- Unbekannte Namen
- Viele Abkürzungen
- `*isk = inet_sk(sk);`
- Selten Kommentare
- Komische C Sachen

Warum eigentlich?

```
1   int ping_recvmsg(struct sock *sk, struct msghdr *msg, size_t len, int flags,
2           int *addr_len)
3   {
4       struct inet_sock *isk = inet_sk(sk);
5       int family = sk->sk_family;
6       struct sk_buff *skb;
7       int copied, err;
8
9       pr_debug("ping_recvmsg(sk=%p,sk->num=%u)\n", isk, isk->inet_num);
10
11
12
13
14
15
16
17
18      skb = skb_recv_datagram(sk, flags, &err);
19      if (!skb)
20          goto out;
```

Flags gesetzt?

- **MSG_OOB**
- **MSG_ERRQUEUE**

- Unbekannte Namen
- Viele Abkürzungen
- `*isk = inet_sk(sk);`
- Selten Kommentare
- Komische C Sachen

Warum eigentlich?

```
1   int ping_recvmsg(struct sock *sk, struct msghdr *msg, size_t len, int flags,
2           int *addr_len)
3   {
4       struct inet_sock *isk = inet_sk(sk);
5       int family = sk->sk_family;
6       struct sk_buff *skb;
7       int copied, err;
8
9       pr_debug("ping_recvmsg(sk=%p,sk->num=%u)\n", isk, isk->inet_num);
10
11      err = -EOPNOTSUPP;
12      if (flags & MSG_OOB)
13          goto out;
14
15      if (flags & MSG_ERRQUEUE)
16          return inet_recv_error(sk, msg, len, addr_len);
17
18      [illegible]
19
20          [illegible]
```

Paket lesen

- Unbekannte Namen
- Viele Abkürzungen
- `*isk = inet_sk(sk);`
- Selten Kommentare
- Komische C Sachen

Warum eigentlich?

Wissen über Grundkonzepte ermöglicht einen (schnellen) Überblick!

- Linux Kernel-Code lesbar ohne sich direkt mit C zu verzetteln

C-ing

- Möglicher Einstieg in den Linux-Kernel

- Vorgehen
  - Bekannte Muster finden
  - Code abstrahieren
  - Interessante Stellen heraussuchen
  - Weiterforschen

Eine Slide zu C

(Ganz kurz. Versprochen!)

Eine Slide zu C

- Struct definiert Datenstrukturen

```
struct myStruct strct;
```

- Ist uns erstmal egal :)

```
strct.value;
strct_ptr->value;
```

- Goto springt

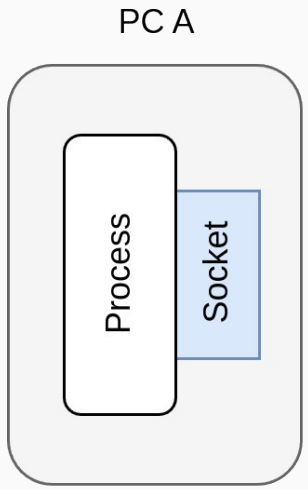
```
if (!dev_valid_name(parms->name))
        goto failed;
```

```
failed:
        return ERR_PTR(err);
```

Pakete im Linux Kernel

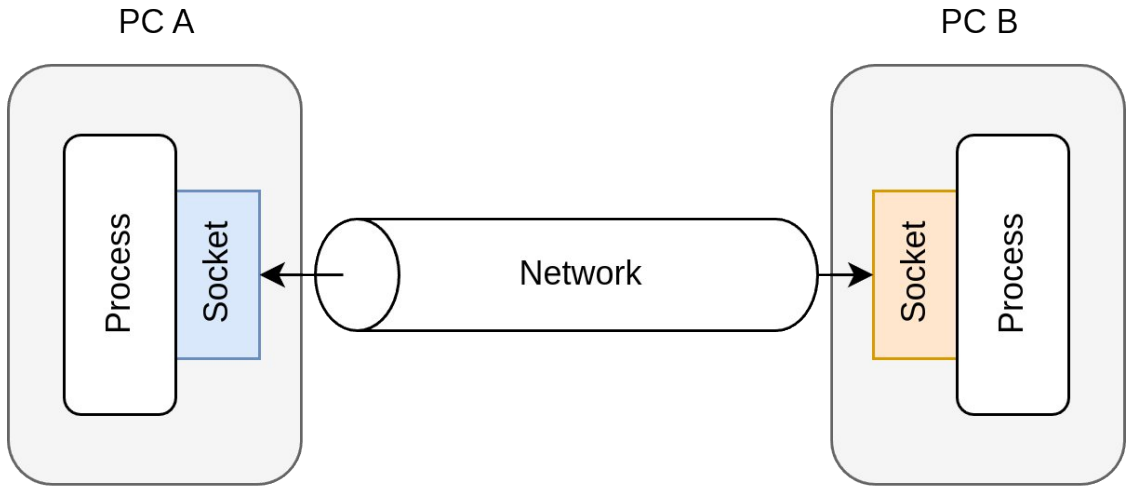
Sockets

- Interface für Kommunikation
  - Prozess erstellt Socket



Sockets

- Interface für Kommunikation
  - Prozess erstellt Socket
  - Verbindung mit anderer Socket
  - Pakete lesen/schreiben
- Kernel managed Verbindung
- Halten Daten die für Verbindung relevant sind
  - Puffer für Pakete (Senden, Empfangen)
  - Quell-/Zieladresse
  - Zustand der Verbindung



Sockets

- Erkennbar an `**sock**`, `**sk**`

```
int ping_recvmsg(struct sock *sk,
int family = sk->sk_family;
```

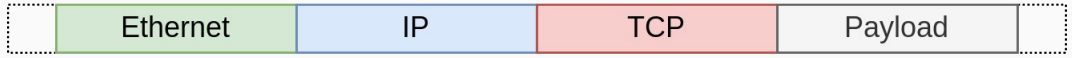
- Protokoll-spezifischer Zugriff

```
struct tcp_sock *tp = tcp_sk(sk);
struct inet_sock *inet = inet_sk(sk);
```

- `**family**` definiert Socket Typ
  - AF_INET für IPv4
  - AF_INET6 für IPv6
  - …

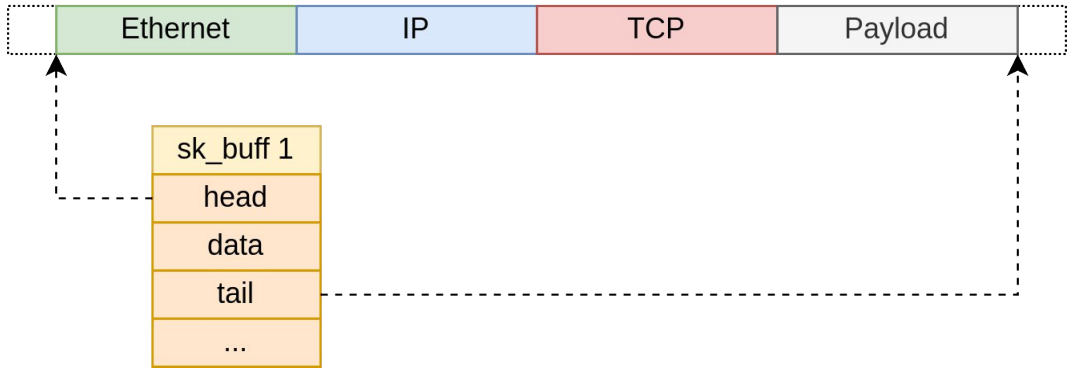
Socket Buffer

- Repräsentiert Pakete im Kernel
- Metadaten-Struktur



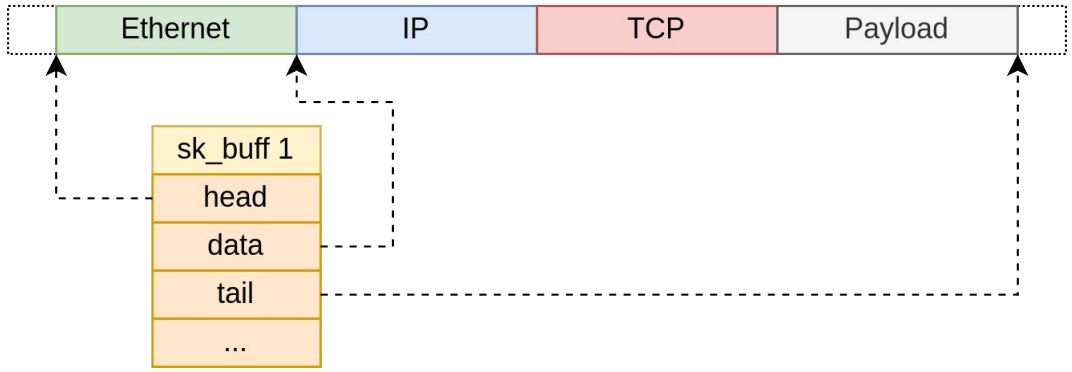
Socket Buffer

- Repräsentiert Pakete im Kernel
- Metadaten-Struktur
  - Pointer zu Daten in Memory



Socket Buffer

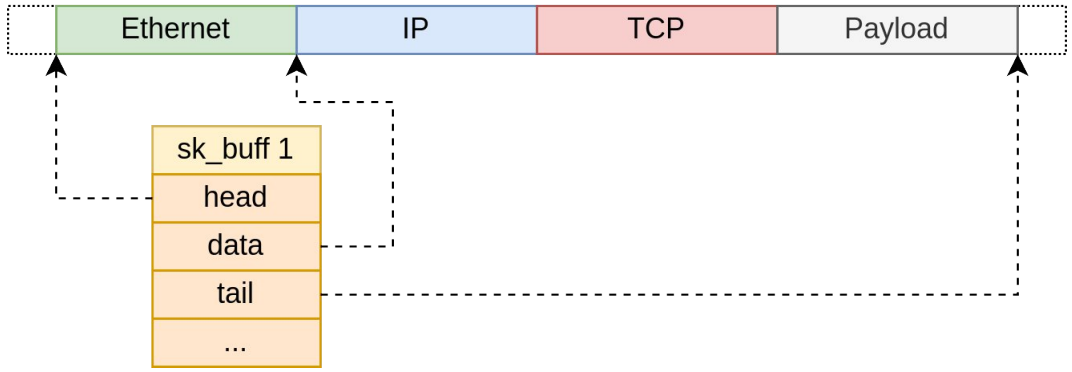
- Repräsentiert Pakete im Kernel
- Metadaten-Struktur
  - Pointer zu Daten in Memory



Socket Buffer

- Repräsentiert Pakete im Kernel

- Metadaten-Struktur
  - Pointer zu Daten in Memory
  - Timestamp
  - Zugehöriges Netzwerk-Device
  - Per-packet Daten in storage `**cb**`

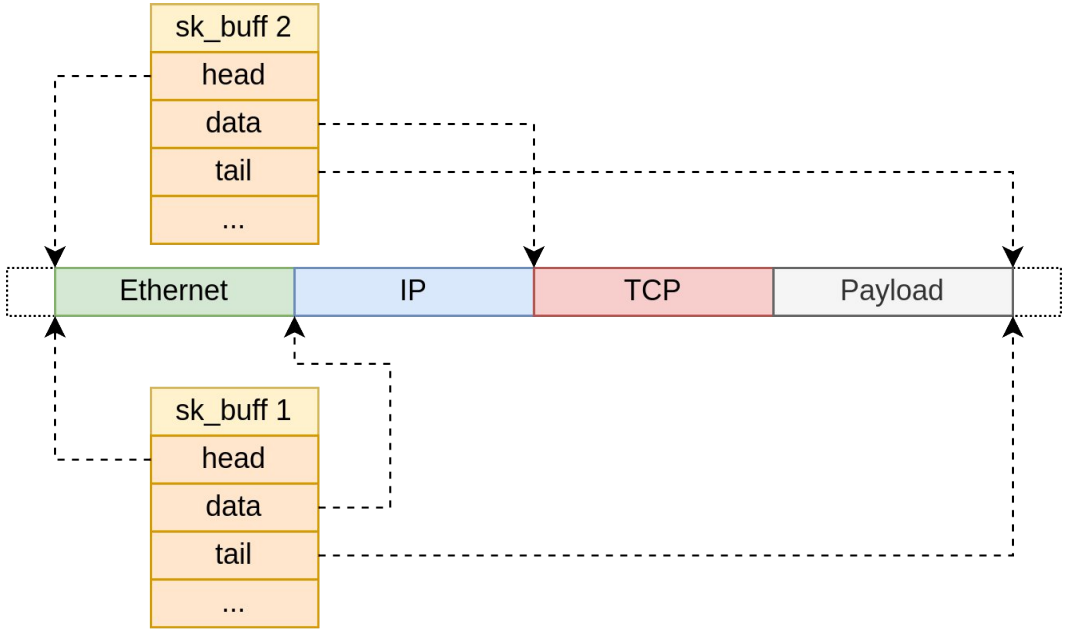


Socket Buffer

- Repräsentiert Pakete im Kernel

- Metadaten-Struktur
  - Pointer zu Daten in Memory
  - Timestamp
  - Zugehöriges Netzwerk-Device
  - Per-packet Daten in storage `**cb**`

- Mehrere *skbs* pro Paket möglich
  - Head und Tail pointer gleich
  - Data pointer zu anderen Stellen



Aus: The Path of a Packet Through the Linux Kernel – Alexander Stephan, Lars Wüstrich

Socket Buffer

- Erkennbar an `**sk_buff**`, `**skb**`

```
struct sk_buff *skb
skb->pkt_type
```

- Zugriff auf Header

```
ip_hdr(skb)->ttl
tcp_hdr(skb)
```

- Zugriff auf per-Packet Daten

```
IPCB(skb)->frag_max_size
TCP_SKB_CB(skb)->seq
```

Flags

- Flags sind True/False Optionen z.B. bei einem Paket
- Mögliche Flags werden als globale Konstanten deklatiert
  - MSG_OOB
  - MSG_ERRQUEUE
  - RTNH_F_LINKDOWN
  - …
- Check ob eine Flag gesetzt wurde

```
if (flags & MSG_OOB)
```

- Eine Flag setzen

```
msg->msg_flags |= MSG_OOB;
```

Flags

GNU Wget
https://www.gnu.org › software › libc › manual › html_node › Out_002dof_002dBand-Data.html

Out-of-Band Data (The GNU C Library)

To send out-of-band data use send, specifying the flag **MSG_OOB** (see Sending Data). Out-of-band data are received with higher priority because the receiving process need not read it in sequence; to read the next available out-of-band data, use recv with the **MSG_OOB** flag (see Receiving Data).

Signal Handling

24 Signal Handling. A signal is a software interrupt delivered to a...

Erste Zusammenfassung

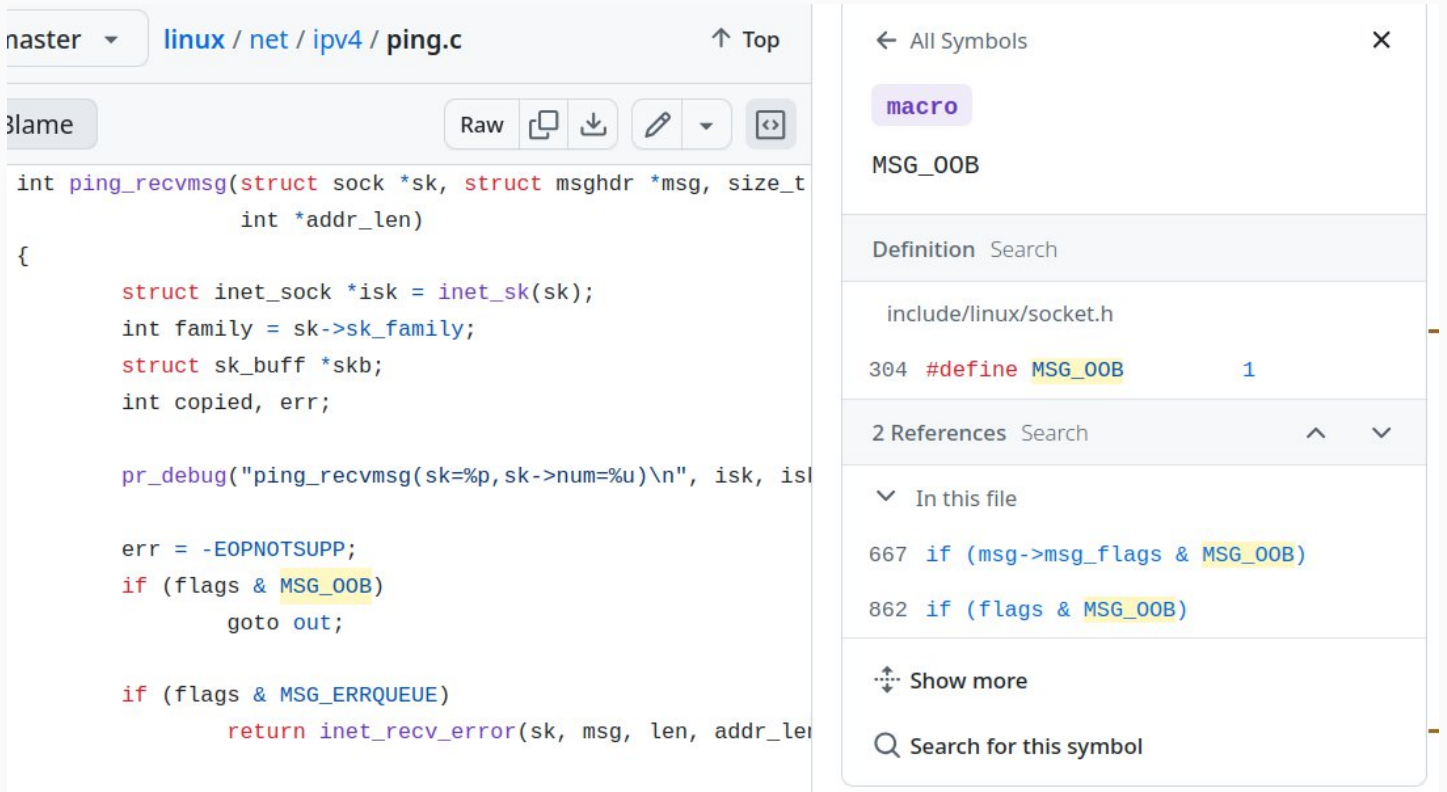
Pattern	Bedeutung
sock *sk*	Verbindungsendpunkt
sk_buff *skb*	Paket
flags & FLAGNAME	Check ob flag FLAGNAME gesetzt
flags |= FLAGNAME	Flag FLAGNAME setzen
cb im Zusammenhang mit *skb*	Per-Packet Daten

- Name von Funktionen sind sehr aussagekräftig
- Was wir nicht verstehen wird erstmal ignoriert

Anwendung

Anwendung

- Source code anschauen
  - Clonen (10 GB)
  - Direkt in Github
- Oberverzeichnis `net/`
  - net/ipv4/…
  - net/mpls/…
- Dateien nach Namen wählen
  - net/ipv4/ping.c
  - net/mpls/af_mpls.c



Anwendung – net/ipv4/ip_foward.c

```
83      int ip_forward(struct sk_buff *skb)
84      {
85              u32 mtu;
86              struct iphdr *iph;      /* Our header */
87              struct rtable *rt;      /* Route we use */
88              struct ip_options *opt  = &(IPCB(skb)->opt);
89              struct net *net;
90              SKB_DR(reason);
91
92              /* that should never happen */
93              if (skb->pkt_type != PACKET_HOST)
94                      goto drop;
95
96              if (unlikely(skb->sk))
97                      goto drop;
```

Anwendung – net/ipv4/ip_foward.c

```
83      int ip_forward(struct sk_buff *skb)
84      {
85              u32 mtu;
86              struct iphdr *iph;      /* Our header */
87              struct rtable *rt;      /* Route we use */
88              struct ip_options *opt  = &(IPCB(skb)->opt);
89              struct net *net;
90              SKB_DR(reason);
91
92              /* that should never happen */
93              if (skb->pkt_type != PACKET_HOST)
94                      goto drop;
95
96              if (unlikely(skb->sk))
97                      goto drop;
```

Route zur Weiterleitung

Anwendung – net/ipv4/ip_foward.c

```
83      int ip_forward(struct sk_buff *skb)
84      {
85              u32 mtu;
86              struct iphdr *iph;      /* Our header */
87              struct rtable *rt;      /* Route we use */
88              struct ip_options *opt  = &(IPCB(skb)->opt);
89              struct net *net;
90              SKB_DR(reason);
91
92              /* that should never happen */
93              if (skb->pkt_type != PACKET_HOST)
94                      goto drop;
95
96              if (unlikely(skb->sk))
97                      goto drop;
```

IP Infos im cb des Pakets

Anwendung – net/ipv4/ip_foward.c

```
87          struct rtable *rt;      /* Route we use */
88          struct ip_options *opt  = &(IPCB(skb)->opt);
91
92          /* that should never happen */
93          if (skb->pkt_type != PACKET_HOST)
94                  goto drop;
```

Paket auf L2 adressiert an diesen Host

Anwendung – net/ipv4/ip_foward.c

```
87          struct rtable *rt;          /* Route we use */
88          struct ip_options *opt   = &(IPCB(skb)->opt);
91
92          /* that should never happen */
93          if (skb->pkt_type != PACKET_HOST)
94                  goto drop;
95
96          if (unlikely(skb->sk))
97                  goto drop;
```

Paket ist einer Socket zugeordnet

Anwendung – net/ipv4/ip_foward.c

Time To Live (TTL) abgelaufen

```
118             if (ip_hdr(skb)->ttl <= 1)
119                     goto too_many_hops;
...
173     too_many_hops:
174             /* Tell the sender its packet died... */
175             __IP_INC_STATS(net, IPSTATS_MIB_INHDRERRORS);
176             icmp_send(skb, ICMP_TIME_EXCEEDED, ICMP_EXC_TTL, 0);
177             SKB_DR_SET(reason, IP_INHDR);
```

Anwendung – net/ipv4/ip_foward.c

```
118             if (ip_hdr(skb)->ttl <= 1)
119                     goto too_many_hops;
...
173     too_many_hops:
174             /* Tell the sender its packet died... */
175             __IP_INC_STATS(net, IPSTATS_MIB_INHDRERRORS);
176             icmp_send(skb, ICMP_TIME_EXCEEDED, ICMP_EXC_TTL, 0);
177             SKB_DR_SET(reason, IP_INHDR);
```

SNMP
Statistik erheben

TTL auf 0
ICMP Benachrichtigung an
ursprünglichen Sender

Anwendung – net/ipv4/ip_foward.c

```
131             __IP_INC_STATS(net, IPSTATS_MIB_OUTFORWDATAGRAMS);
132
133             IPCB(skb)->flags |= IPSKB_FORWARDED;
134             mtu = ip_dst_mtu_maybe_forward(&rt->dst, true);
135             if (ip_exceeds_mtu(skb, mtu)) {
136                     IP_INC_STATS(net, IPSTATS_MIB_FRAGFAILS);
137                     icmp_send(skb, ICMP_DEST_UNREACH, ICMP_FRAG_NEEDED,
138                               htonl(mtu));
139                     SKB_DR_SET(reason, PKT_TOO_BIG);
140                     goto drop;
141             }
```

Anwendung – net/ipv4/ip_foward.c

Statistik erheben (wieder SNMP)

```
131             __IP_INC_STATS(net, IPSTATS_MIB_OUTFORWDATAGRAMS);
132
133             IPCB(skb)->flags |= IPSKB_FORWARDED;
134             mtu = ip_dst_mtu_maybe_forward(&rt->dst, true);
135             if (ip_exceeds_mtu(skb, mtu)) {
136                     IP_INC_STATS(net, IPSTATS_MIB_FRAGFAILS);
137                     icmp_send(skb, ICMP_DEST_UNREACH, ICMP_FRAG_NEEDED,
138                               htonl(mtu));
139                     SKB_DR_SET(reason, PKT_TOO_BIG);
140                     goto drop;
141             }
```

Anwendung – net/ipv4/ip_foward.c

```
131             __IP_INC_STATS(net, IPSTATS_MIB_OUTFORWDATAGRAMS);
132
133             IPCB(skb)->flags |= IPSKB_FORWARDED;
134             mtu = ip_dst_mtu_maybe_forward(&rt->dst, true);
135             if (ip_exceeds_mtu(skb, mtu)) {
136                     IP_INC_STATS(net, IPSTATS_MIB_FRAGFAILS);
137                     icmp_send(skb, ICMP_DEST_UNREACH, ICMP_FRAG_NEEDED,
138                               htonl(mtu));
139                     SKB_DR_SET(reason, PKT_TOO_BIG);
140                     goto drop;
141             }
```

Flag setzen

Anwendung – net/ipv4/ip_foward.c

```
131             __IP_INC_STATS(net, IPSTATS_MIB_OUTFORWDATAGRAMS);
132
133             IPCB(skb)->flags |= IPSKB_FORWARDED;
134             mtu = ip_dst_mtu_maybe_forward(&rt->dst, true);
135             if (ip_exceeds_mtu(skb, mtu)) {
136                     IP_INC_STATS(net, IPSTATS_MIB_FRAGFAILS);
137                     icmp_send(skb, ICMP_DEST_UNREACH, ICMP_FRAG_NEEDED,
138                               htonl(mtu));
139                     SKB_DR_SET(reason, PKT_TOO_BIG);
140                     goto drop;
141             }
```

MTU für nächsten Hop holen

Anwendung – net/ipv4/ip_foward.c

```
131             __IP_INC_STATS(net, IPSTATS_MIB_OUTFORWDATAGRAMS);
132
133             IPCB(skb)->flags |= IPSKB_FORWARDED;
134             mtu = ip_dst_mtu_maybe_forward(&rt->dst, true);
135             if (ip_exceeds_mtu(skb, mtu)) {
136                     IP_INC_STATS(net, IPSTATS_MIB_FRAGFAILS);
137                     icmp_send(skb, ICMP_DEST_UNREACH, ICMP_FRAG_NEEDED,
138                               htonl(mtu));
139                     SKB_DR_SET(reason, PKT_TOO_BIG);
140                     goto drop;
141             }
```

Paket zu groß?

Anwendung – net/ipv4/ip_foward.c

```
131             __IP_INC_STATS(net, IPSTATS_MIB_OUTFORWDATAGRAMS);
132
133             IPCB(skb)->flags |= IPSKB_FORWARDED;
134             mtu = ip_dst_mtu_maybe_forward(&rt->dst, true);
135             if (ip_exceeds_mtu(skb, mtu)) {
136                     IP_INC_STATS(net, IPSTATS_MIB_FRAGFAILS);
137                     icmp_send(skb, ICMP_DEST_UNREACH, ICMP_FRAG_NEEDED,
138                               htonl(mtu));
139                     SKB_DR_SET(reason, PKT_TOO_BIG);
140                     goto drop;
141             }
```

ICMP Benachrichtigung an unrsprünglichen Sender

Drop Reason wird gesetzt?

Paket droppen

Danke fürs Zuhören!

```
/* Fuck, we are miserable poor guys... */
```

```
/* Misery. We are in troubles, going to mincer fragments... */
```

```
/* #define SOL_ICMP     1       No-no-no! Due to Linux :-) we cannot use SOL_ICMP=1 */
```

```
/* 2. Fixups made earlier cannot be right.
 *    If we do not estimate RTO correctly without them,
 *    all the algo is pure shit and should be replaced
 *    with correct one. It is exactly, which we pretend to do.
 */
```

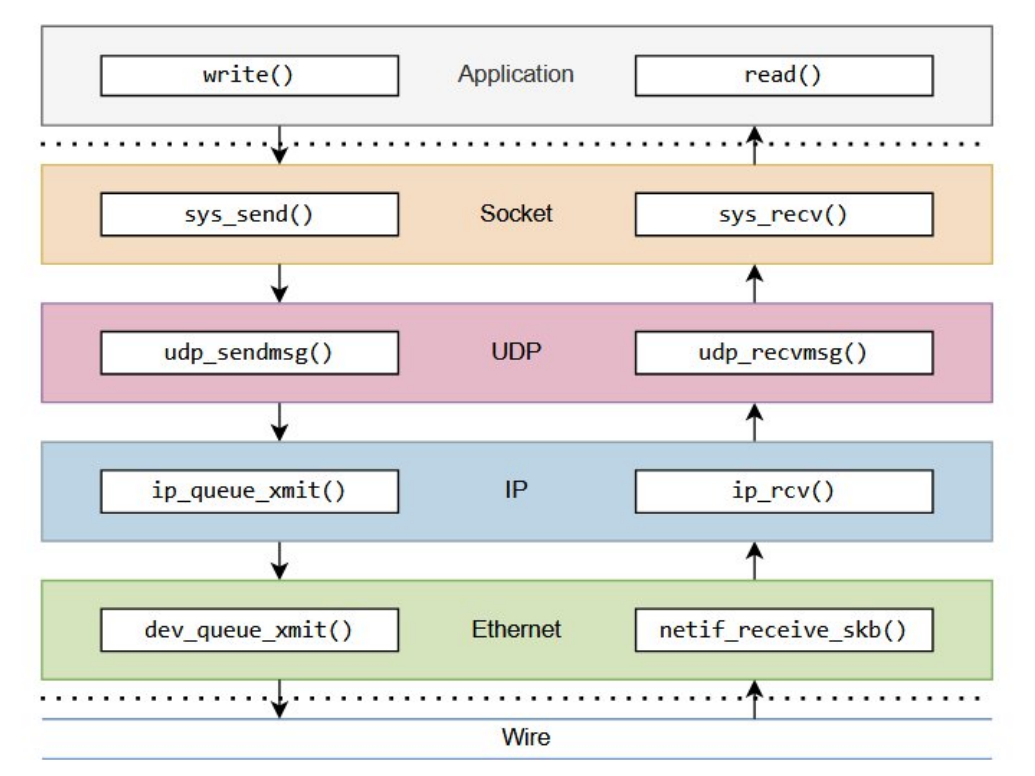
```
PLEASE, do not touch this function. If you think, that it is
incorrect, grep kernel sources and think about consequences
before trying to improve it.
```

```
static bool arp_is_garp(
```

Weiterlesen

The Path of a Packet Through the Linux Kernel

- Alexander Stephan, Lars Wüstrich (2024)



Extra – tcp_sk

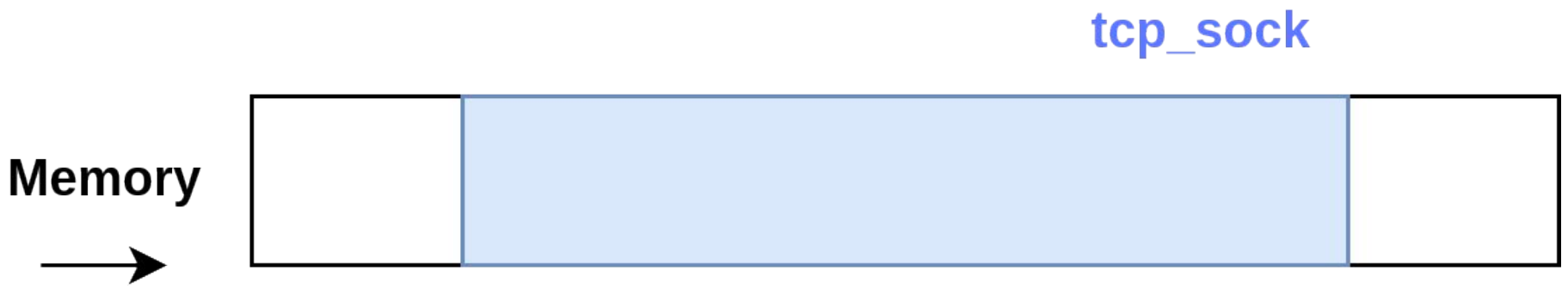
```
struct tcp_sock *tp = tcp_sk(sk);
```

WIE ????

Extra – tcp_sk

```
struct tcp_sock {
        /* Cacheline organization can be found documented in
         * Documentation/networking/net_cachelines/tcp_sock.rst.
         * Please update the document when adding new fields.
         */

        /* inet_connection_sock has to be the first member of tcp_sock */
        struct inet_connection_sock     inet_conn;
```

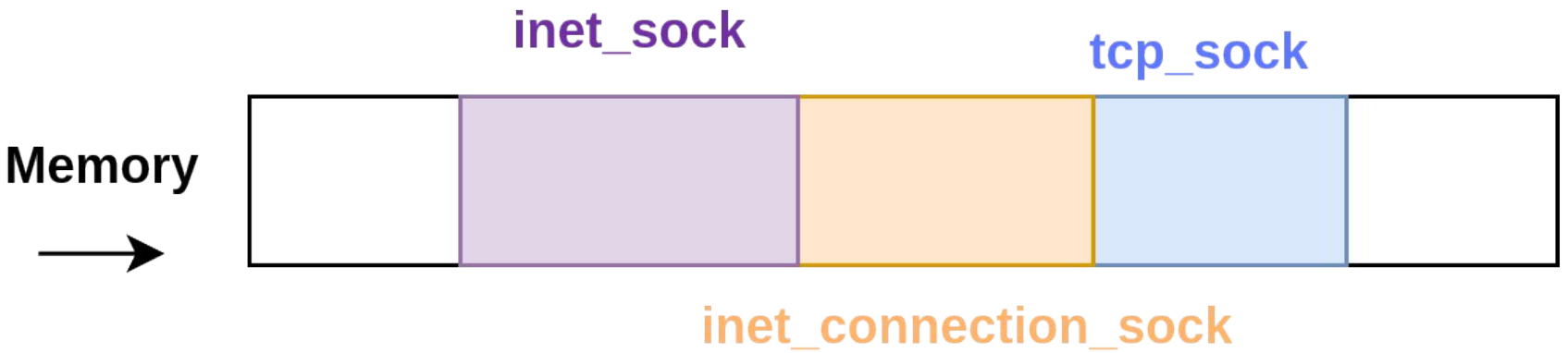


Extra – tcp_sk

```
78  struct inet_connection_sock {
79          /* inet_sock has to be the first member! */
80          struct inet_sock          icsk_inet;
```

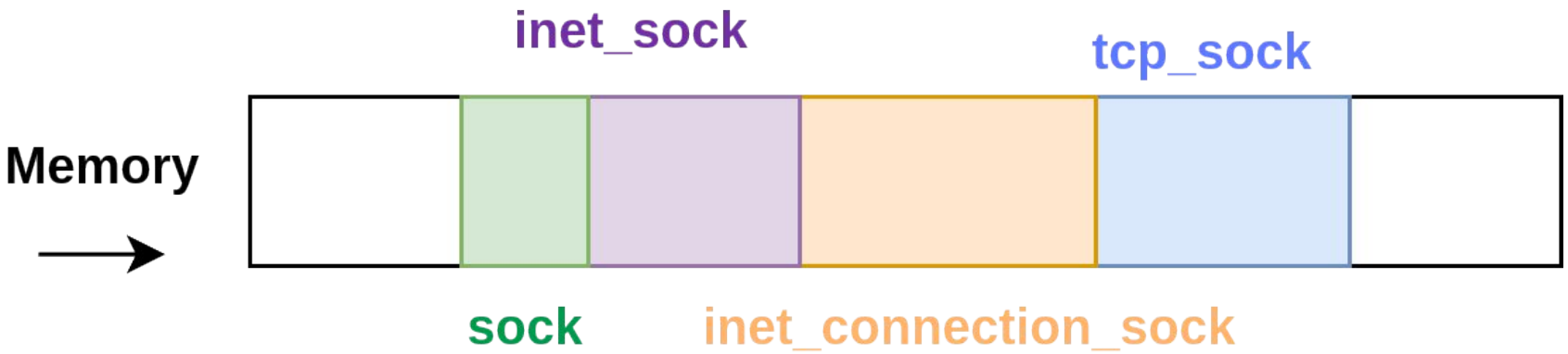
Extra – tcp_sk

```
212  struct inet_sock {
213          /* sk and pinet6 has to be the first two members of inet_sock */
214          struct sock             sk;
```



Extra – tcp_sk

```
struct tcp_sock *tp = tcp_sk(sk);
```



Extra – net/ipv4/ip_foward.c

```
118             if (ip_hdr(skb)->ttl <= 1)
119                     goto too_many_hops;
120
121             if (!xfrm4_route_forward(skb)) {
122                     SKB_DR_SET(reason, XFRM_POLICY);
123                     goto drop;
124             }
125
126             rt = skb_rtable(skb);
127
128             if (opt->is_strictroute && rt->rt_uses_gateway)
129                     goto sr_failed;
```

Time To Live (TTL) abgelaufen

Hier müsste man mal ip XFRM suchen …

Route wählen

Strict Routing und Route geht durch Gateway

Extra– net/ipv4/ip_foward.c


```
 99          if (skb_warn_if_lro(skb))
100                  goto drop;
101
102          if (!xfrm4_policy_check(NULL, XFRM_POLICY_FWD, skb)) {
103                  SKB_DR_SET(reason, XFRM_POLICY);
104                  goto drop;
105          }
106
107          if (IPCB(skb)->opt.router_alert && ip_call_ra_chain(skb))
108                  return NET_RX_SUCCESS;
109
110          skb_forward_csum(skb);
111          net = dev_net(skb->dev);
```

Extra – net/mpls/af_mpls.c

```
360     mdev = mpls_dev_get(dev);
361     if (!mdev)
362             goto drop;
363
364     MPLS_INC_STATS_LEN(mdev, skb->len, rx_packets,
365                        rx_bytes);
366
367     if (!mdev->input_enabled) {
368             MPLS_INC_STATS(mdev, rx_dropped);
369             goto drop;
370     }
371
372     if (skb->pkt_type != PACKET_HOST)
373             goto err;
```

- Check ob MPLS Net Device verfügbar
- Check ob kein Input Device
- Wenn paket nicht für den lokalen Host

Extra 1

```
if (nh_flags & (RTNH_F_DEAD | RTNH_F_LINKDOWN))
        continue;
```

Extra 1

```
/* Remove any debris in the socket control block */
memset(IPCB(skb), 0, sizeof(struct inet_skb_parm));
IPCB(skb)->iif = skb->skb_iif;
```