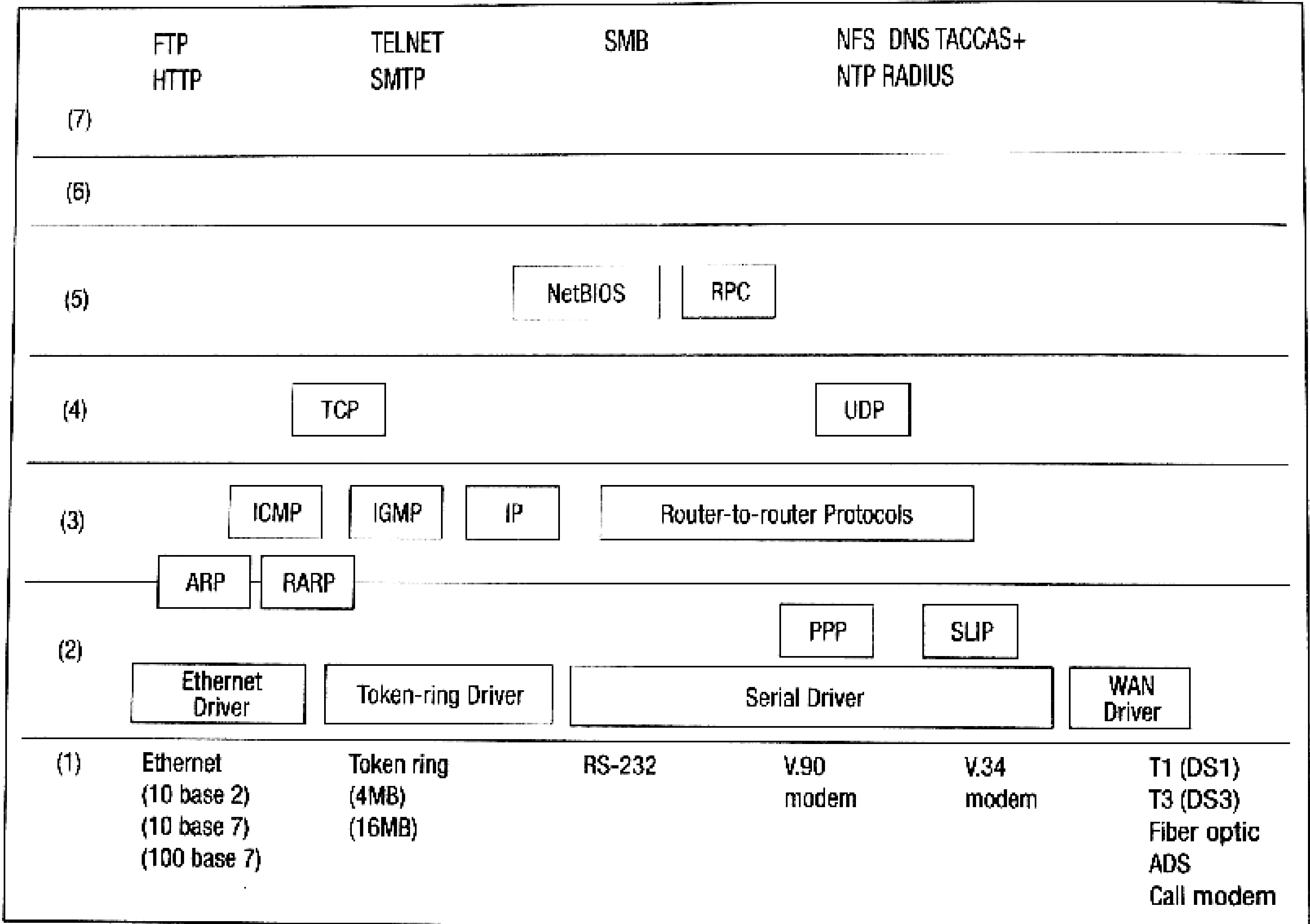
Linux IP Networking

Tsai Ming-Yao
NCTU CIS Computer System Lab.
2002 Linux Kernel Trace Seminar
Aug. 26, 2002

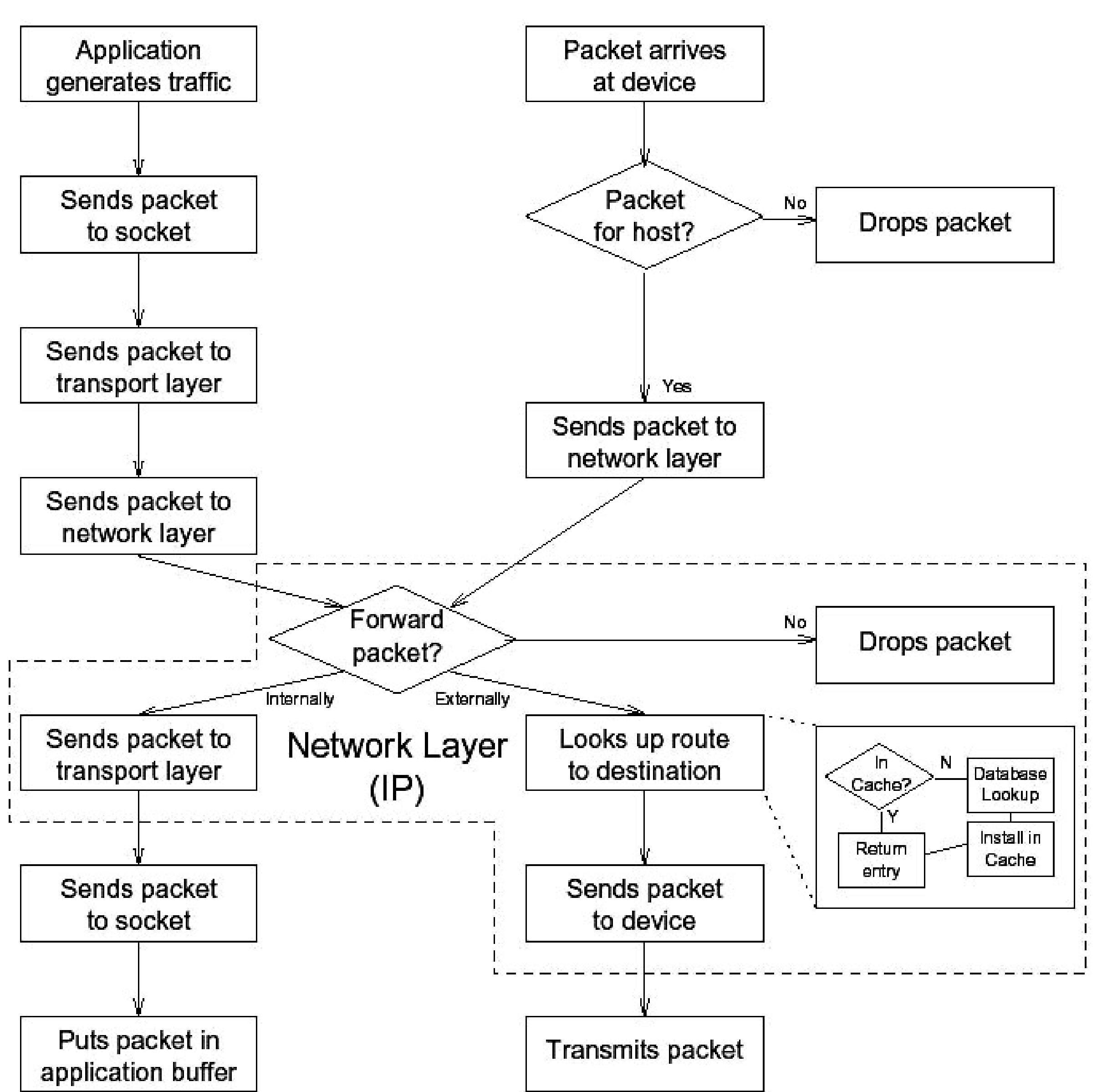
Outline

1. Introduction
2. Connections
3. Sending Messages
4. Receiving Messages
5. IP Forwarding
6. Basic Internet Protocol Routing

TCP/IP Protocols in ISO model



Abstraction of the Linux message traffic path.



Packet Structure

- A socket buffer
  - sk_buff
- Payload data is copied only twice
  - user space to kernel space
  - kernel space to output medium
- All protocol use the same sk_buff

Field	Description
sk	pointer to owning socket
stamp	arrival time
dev	pointer to receiving/transmitting device
h	pointer to transport layer header
nh	pointer to network layer header
mac	pointer to link layer header
dst	pointer to dst_entry
cb	TCP per-packet control information
len	actual data length
csum	checksum
protocol	packet network protocol
truesize	buffer size
head	pointer to head of buffer
data	pointer to data head
tail	pointer to tail
end	pointer to end
destructor	pointer to destruct function

include/linux/skbuff.h

struct sk_buff{}

```
/* These two members must be first. */
struct sk_buff          * next;         /* Next buffer in list                    */
struct sk_buff          * prev;         /* Previous buffer in list                */

struct sk_buff_head     * list;         /* List we are on                         */
struct sock             *sk;            /* Socket we are owned by                 */
struct timeval          stamp;          /* Time we arrived                        */
struct net_device       *dev;           /* Device we arrived on/are leaving by    */
……………………………………………………………
……………………………………………………………
unsigned short          protocol;       /* Packet protocol from driver.           */
unsigned short          security;       /* Security level of packet               */
unsigned int            truesize;       /* Buffer size                            */

unsigned char           *head;          /* Head of buffer                         */
unsigned char           *data;          /* Data head pointer                      */
unsigned char           *tail;          /* Tail pointer                           */
unsigned char           *end;           /* End pointer                            */

void      (*destructor)(struct sk_buff *);   /* Destruct function   */
```

1. Introduction
2. **Connections**
3. Sending Messages
4. Receiving Messages
5. IP Forwarding
6. Basic Internet Protocol Routing

Connection Processes in IP

1. Check for errors
2. Determine route to destination:
   - Check routing table for existing entry
   - Look up destination in FIB (Forwarding Information Base)
   - Build new routing table entry
   - Put entry in routing table and return it
3. Store pointer to routing entry in socket
4. Call protocol specific connection function (ex:TCP)
5. Set socket state to established

net/ipv4/tcp_ipv4.c

tcp_v4_connect() (1)

```
int tcp_v4_connect(struct sock *sk, struct sockaddr *uaddr, int addr_len)
{…………………………
    if (addr_len < sizeof(struct sockaddr_in))
            return(-EINVAL);

    if (usin->sin_family != AF_INET)
            return(-EAFNOSUPPORT);

    nexthop = daddr = usin->sin_addr.s_addr;

    if (sk->protinfo.af_inet.opt && sk->protinfo.af_inet.opt->srr) {
        if (daddr == 0)
                return -EINVAL;
        nexthop = sk->protinfo.af_inet.opt->faddr;
    }
…………………………
}
```

This will initialize an outgoing connection

Check for errors

net/ipv4/tcp_ipv4.c

tcp_v4_connect() (2)

```
int tcp_v4_connect(struct sock *sk, struct sockaddr *uaddr, int addr_len)
{............................
 ............................
     tmp = ip_route_connect(&rt, nexthop, sk->saddr,
                     RT_CONN_FLAGS(sk), sk->bound_dev_if);

     if (tmp < 0)
            return tmp;

     if (rt->rt_flags&(RTCF_MULTICAST|RTCF_BROADCAST)) {
         ip_rt_put(rt);
         return -ENETUNREACH;
     }
............................
............................
}
```

determine route to destination

Network Unreachable

net/ipv4/tcp_ipv4.c

tcp_v4_connect() (3)

```
int tcp_v4_connect(struct sock *sk, struct sockaddr *uaddr, int addr_len)
{…………………………
      tcp_set_state(sk, TCP_SYN_SENT);
      err = tcp_v4_hash_connect(sk);
      if (err)
               goto failure;

      if (!tp->write_seq)
        tp->write_seq = secure_tcp_sequence_number(sk->saddr, sk->daddr, \
                                                   sk->sport, usin->sin_port);
      sk->protinfo.af_inet.id = tp->write_seq^jiffies;

      err = tcp_connect(sk);
      if (err)
        goto failure;
      return 0;
}
```

protocol specific connection function

```
failure:
        tcp_set_state(sk, TCP_CLOSE);
        __sk_dst_reset(sk);
        sk->route_caps = 0;
        sk->dport = 0;
        return err;
```

net/ipv4/tcp_output.c

tcp_connect()

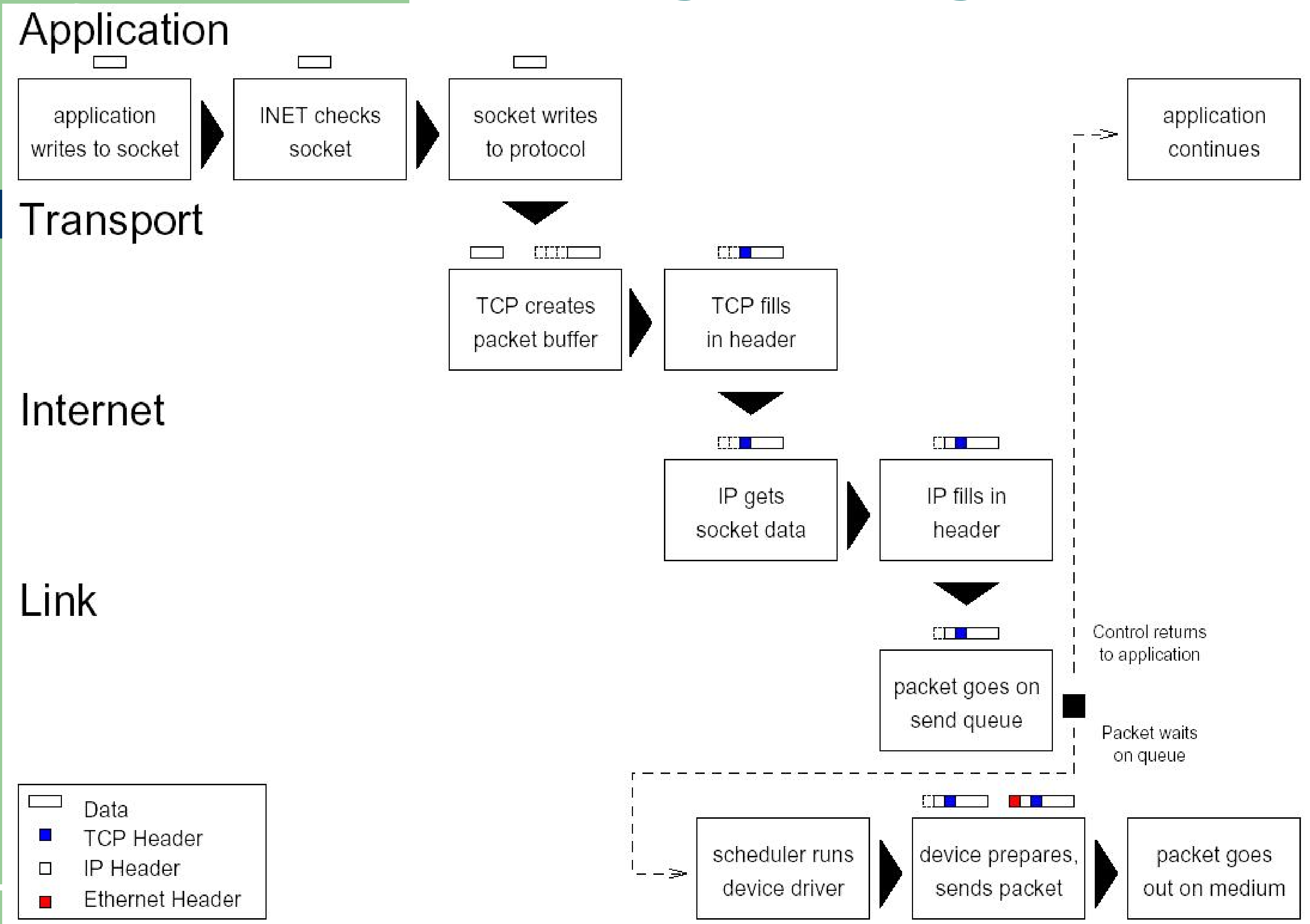
```
int tcp_connect(struct sock *sk)
{........................
        tcp_connect_init(sk);
        buff = alloc_skb(MAX_TCP_HEADER + 15, sk->allocation);
        if (unlikely(buff == NULL))      return -ENOBUFS;

        /* Reserve space for headers. */
..........................
        /* Send it off. */
        tcp_charge_skb(sk, buff);
        tp->packets_out++;
        tcp_transmit_skb(sk, skb_clone(buff, GFP_KERNEL));
        TCP_INC_STATS(TcpActiveOpens);

        /* Timer for repeating the SYN until an answer. */
        tcp_reset_xmit_timer(sk, TCP_TIME_RETRANS, tp->rto);
        return 0;
}
```

bits and window size set

Send packet, initiating TCP connection

1. Introduction
2. Connections
3. **Sending Messages**
4. Receiving Messages
5. IP Forwarding
6. Basic Internet Protocol Routing

Sending Messages



Wrapping a Packet in IP

1. Create a packet buffer
2. Look up route to destination
3. Fill in the packet IP header
4. Copy the transport header and the payload from user space
5. Send the packet to the destination route's device output function

net/ipv4/ip_output.c

ip_build_xmit() (1)

```
int ip_build_xmit(struct sock *sk, int getfrag (const void *, char *, unsigned int, \
      unsigned int), const void *frag, unsigned length, struct ipcm_cookie *ipc, \
      struct rtable *rt, int flags)
{.............................................................................
      int hh_len = (rt->u.dst.dev->hard_header_len + 15)&~15;

      skb = sock_alloc_send_skb(sk, length+hh_len+15,
                                flags&MSG_DONTWAIT, &err);
      if(skb==NULL)
              goto error;
      skb_reserve(skb, hh_len);

      skb->priority = sk->priority;
      skb->dst = dst_clone(&rt->u.dst);

      skb->nh.iph = iph = (struct iphdr *)skb_put(skb, length);
}
```

allocate memory for skb

sets up skb header

net/ipv4/ip_output.c

ip_build_xmit() (2)

```
if(!sk->protinfo.af_inet.hdrincl) {
        iph->version=4;
        iph->ihl=5;
        iph->tos=sk->protinfo.af_inet.tos;
        iph->tot_len = htons(length);
        iph->frag_off = df;
        iph->ttl=sk->protinfo.af_inet.mc_ttl;
        ip_select_ident(iph, &rt->u.dst, sk);
        if (rt->rt_type != RTN_MULTICAST)
                iph->ttl=sk->protinfo.af_inet.ttl;
        iph->protocol=sk->protocol;
        iph->saddr=rt->rt_src;
        iph->daddr=rt->rt_dst;
        iph->check=0;
        iph->check = ip_fast_csum((unsigned char *)iph, iph->ihl);
        err = getfrag(frag, ((char *)iph)+iph->ihl*4,0, length-iph->ihl*4);
}
else    err = getfrag(frag, (void *)iph, 0, length);
```

copy buffer from user space

net/ipv4/ip_output.c

ip_queue_xmit() (1)

```
int ip_queue_xmit(struct sk_buff *skb)
{………………
      rt = (struct rtable *)__sk_dst_check(sk, 0);
      if (rt == NULL) {
                u32 daddr;

                daddr = sk->daddr;
                if(opt && opt->srr)
                          daddr = opt->faddr;

                if (ip_route_output(&rt, daddr, sk->saddr,  RT_CONN_FLAGS(sk), \
                                                          sk->bound_dev_if))
                          goto no_route;
                __sk_dst_set(sk, &rt->u.dst);
                sk->route_caps = rt->u.dst.dev->features;
      }
skb->dst = dst_clone(&rt->u.dst);

………………
}
```

Make sure we can route this packet.

Use correct destination address if we have options.

If this fails, retransmit mechanism of transport layer will keep trying until route appears or the connection times itself out.

net/ipv4/ip_output.c

ip_queue_xmit() (2)

```
    /* OK, we know where to send it, allocate and build IP header. */
iph = (struct iphdr *) skb_push(skb, sizeof(struct iphdr) + (opt ? opt->optlen : 0));
*((__u16 *)iph) = htons((4 << 12) | (5 << 8) | (sk->protinfo.af_inet.tos & 0xff));
iph->tot_len = htons(skb->len);

if (ip_dont_fragment(sk, &rt->u.dst))
        iph->frag_off = __constant_htons(IP_DF);
else
        iph->frag_off = 0;

iph->ttl      = sk->protinfo.af_inet.ttl;
iph->protocol = sk->protocol;
iph->saddr    = rt->rt_src;
iph->daddr    = rt->rt_dst;
skb->nh.iph   = iph;
………
return NF_HOOK(PF_INET, NF_IP_LOCAL_OUT, skb, \
            NULL, rt->u.dst.dev, ip_queue_xmit2);
```

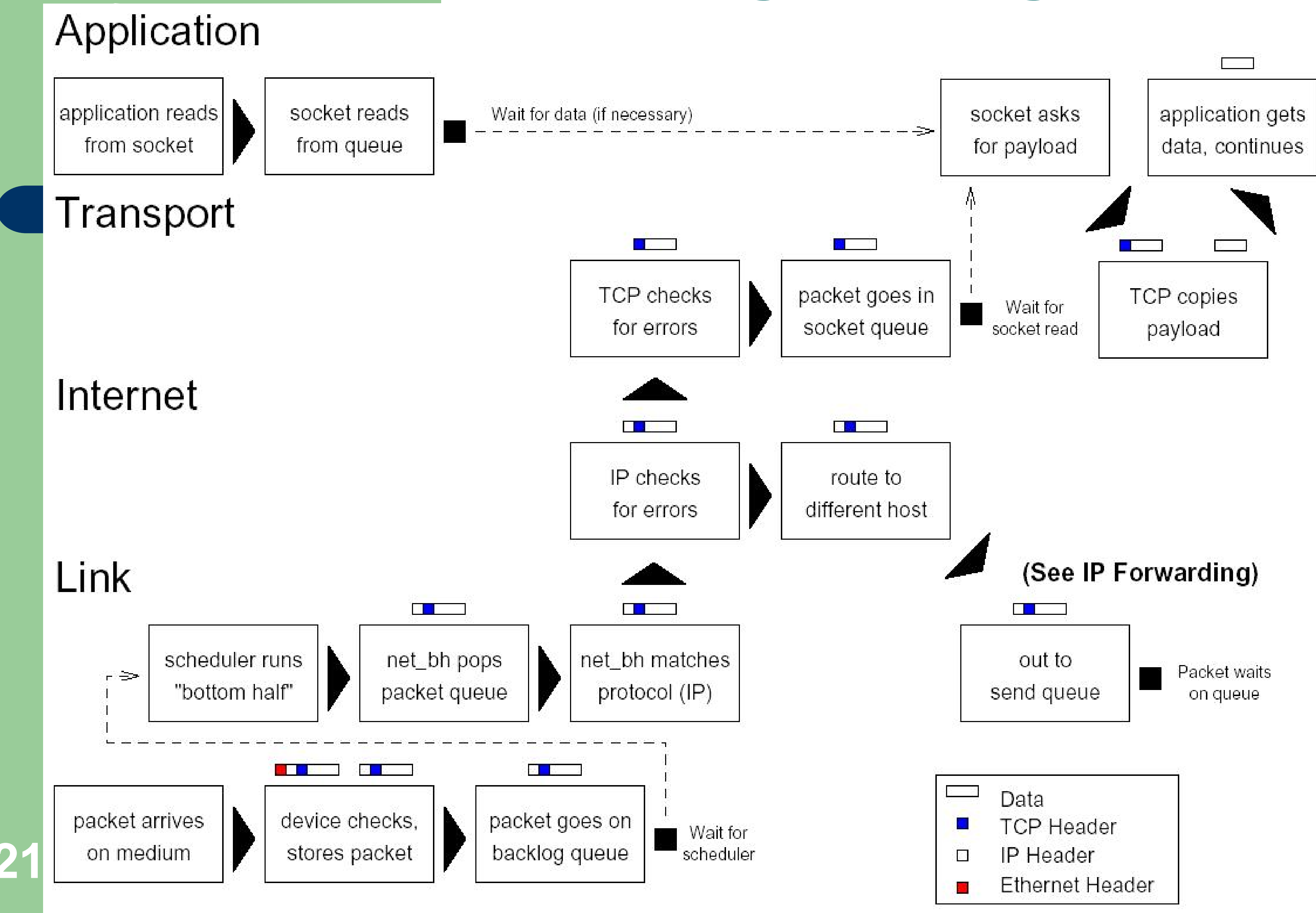
build IP header

fragment if required

adds IP checksum
calls skb->dst->output()

1. Introduction
2. Connections
3. Sending Messages
4. **Receiving Messages**
5. IP Forwarding
6. Basic Internet Protocol Routing

Receiving Messages



Unwrapping a Packet in IP

1. Check packet for errors
   - Too short or long? Invalid version? Checksum error?
2. Defragment the packet if necessary
3. Get the route for the packet
   - For this host? To be forwarded?
4. Send the packet to its destination handling routine
   (TCP or UDP reception, or possibly retransmission to another host)

net/ipv4/ip_input.c

ip_rcv() (1)

```
int ip_rcv(struct sk_buff *skb, struct net_device *dev, struct packet_type *pt)
{............
      if (iph->ihl < 5 || iph->version != 4)
              goto inhdr_error;
      if (!pskb_may_pull(skb, iph->ihl*4))
              goto inhdr_error;

      iph = skb->nh.iph;

      if (ip_fast_csum((u8 *)iph, iph->ihl) != 0)
              goto inhdr_error;
      {
               __u32 len = ntohs(iph->tot_len);
              if (skb->len < len || len < (iph->ihl<<2))
                          goto inhdr_error;
............
}
```

examines packet for errors

net/ipv4/ip_input.c

ip_rcv() (2)

```
		if (skb->len > len) {
			__pskb_trim(skb, len);
			if (skb->ip_summed == CHECKSUM_HW)
				skb->ip_summed = CHECKSUM_NONE;
		}
	}
	return NF_HOOK(PF_INET, NF_IP_PRE_ROUTING, skb, dev, NULL,
			ip_rcv_finish);
inhdr_error:
	IP_INC_STATS_BH(IpInHdrErrors);
drop:
	kfree_skb(skb);
out:
	return NET_RX_DROP;
}
```

remove padding

net/ipv4/ip_input.c

ip_rcv_finish() (1)

```
static inline int ip_rcv_finish(struct sk_buff *skb)
{............
     if (skb->dst == NULL) {
          if (ip_route_input(skb, iph->daddr, iph->saddr, iph->tos, dev))
               goto drop;
     }
 ...............
     if (iph->ihl > 5) {
          struct ip_options *opt;
          ..................
          if (ip_options_compile(NULL, skb))
               goto inhdr_error;
............
}
```

describe how to travel inside networking

examines and handle IP options

net/ipv4/ip_input.c

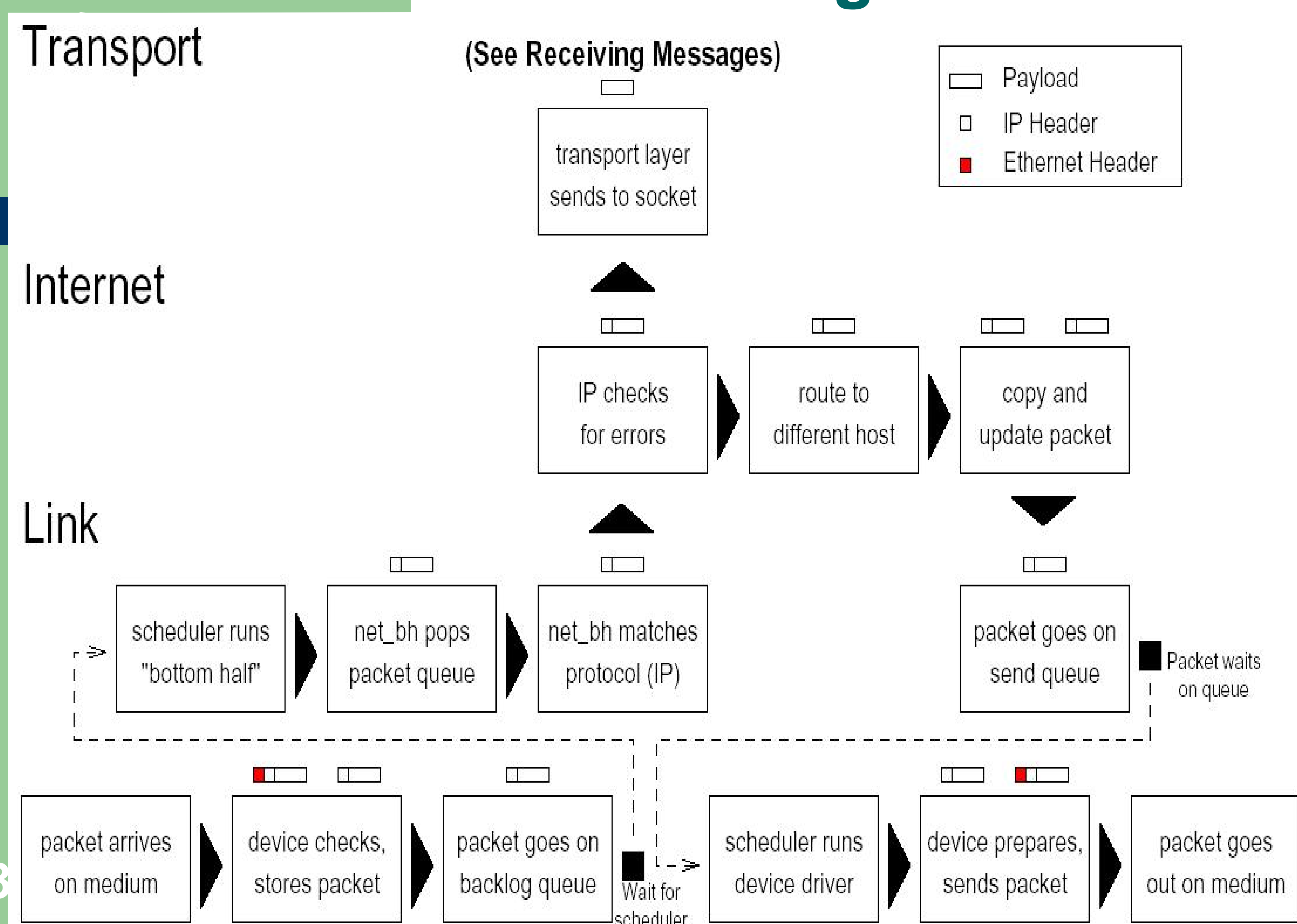
ip_rcv_finish() (2)

```
opt = &(IPCB(skb)->opt);
        if (opt->srr) {
            struct in_device *in_dev = in_dev_get(dev);
            if (in_dev) {
                if (!IN_DEV_SOURCE_ROUTE(in_dev)) {
                    if (IN_DEV_LOG_MARTIANS(in_dev) && net_ratelimit())
                        printk(KERN_INFO "source route option %u.%u.%u.%u -> \
                                                                    %u.%u.%u.%u\n",
                            NIPQUAD(iph->saddr), NIPQUAD(iph->daddr));
                    in_dev_put(in_dev);
                    goto drop;
                }
                in_dev_put(in_dev);
            }
            if (ip_options_rcv_srr(skb))
                goto drop;
        }
    }

    return skb->dst->input(skb);
………..
```

source route option

1. Introduction
2. Connections
3. Sending Messages
4. Receiving Messages
5. **IP Forwarding**
6. Basic Internet Protocol Routing

IP forwarding



Forwarding a Packet in IP

1. Check TTL field (and decreasing it)
2. Check packet for improper routing
3. Send ICMP back to sender if there are any problems
4. Copy packet into new buffer and free old one
5. Set any IP options
6. Fragment packet if it is too big for new destination
7. Send the packet to the destination route's device output function

net/ipv4/ip_forward.c

ip_forward() (1)

```
int ip_forward(struct sk_buff *skb)
{
…………
     if (IPCB(skb)->opt.router_alert && ip_call_ra_chain(skb))
               return NET_RX_SUCCESS;

     if (skb->pkt_type != PACKET_HOST)
               goto drop;
…………
```

checks for router alert

if packet is not meant for any host, drop it!!

net/ipv4/ip_forward.c

ip_forward() (2)

```
/*     According to the RFC, we must first decrease the TTL field. If
 *     that reaches zero, we must reply an ICMP control message telling
 *     that the packet's lifetime expired.
 */
    iph = skb->nh.iph;
    rt = (struct rtable*)skb->dst;

    if (iph->ttl <= 1)
            goto too_many_hops;

    if (opt->is_strictroute && rt->rt_dst != rt->rt_gateway)
            goto sr_failed;
```

if TTL has expired, drop packet and sends ICMP message back

if strict route cannot be followed, drops packet and sends ICMP message back to sender

net/ipv4/ip_forward.c

ip_forward() (3)

```
.........
      /*    We now generate an ICMP HOST REDIRECT giving the route
       *    we calculated.
       */
       if (rt->rt_flags&RTCF_DOREDIRECT && !opt->srr)
              ip_rt_send_redirect(skb);
............
     /* We now may allocate a new buffer, and copy the datagram into it.
      * If the indicated interface is up and running, kick it.
     */
        if (skb->len > mtu && (ntohs(iph->frag_off) & IP_DF))
              goto frag_needed;
.........
 return NF_HOOK(PF_INET, NF_IP_FORWARD, skb, skb->dev, dev2, ip_forward_finish);
```

telling sender packet is redirected copies and release old packet

need to defragment, send ICMP message FRAG_NEEDED back

include/net/ip.h

ip_send()

```
static inline int ip_send(struct sk_buff *skb)
{
        if (skb->len > skb->dst->pmtu)
                return ip_fragment(skb, ip_finish_output);
        else
                return ip_finish_output(skb);
}
```

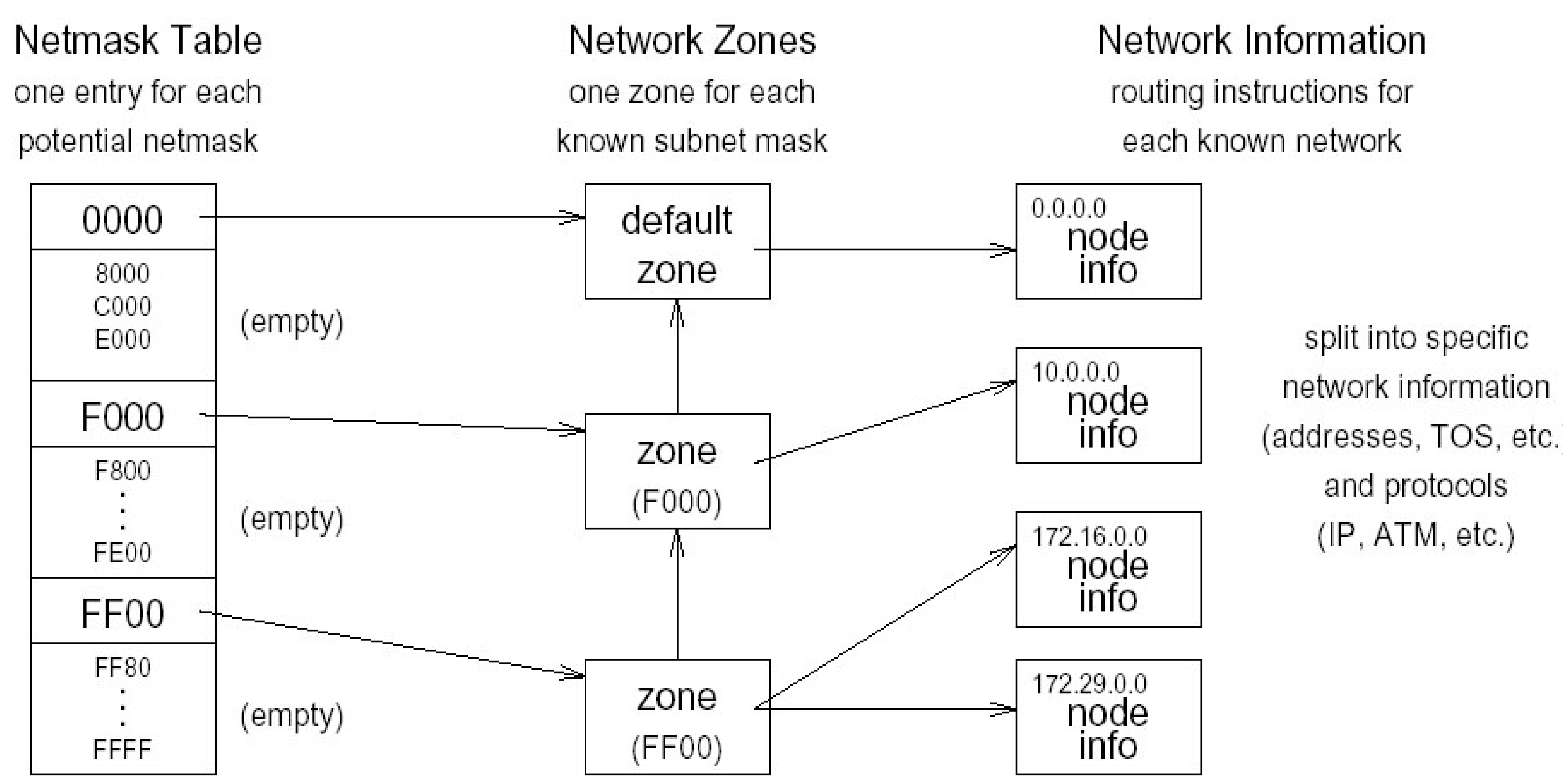
If packet is too big for device

1. Introduction
2. Connections
3. Sending Messages
4. Receiving Messages
5. IP Forwarding
6. **Basic Internet Protocol Routing**

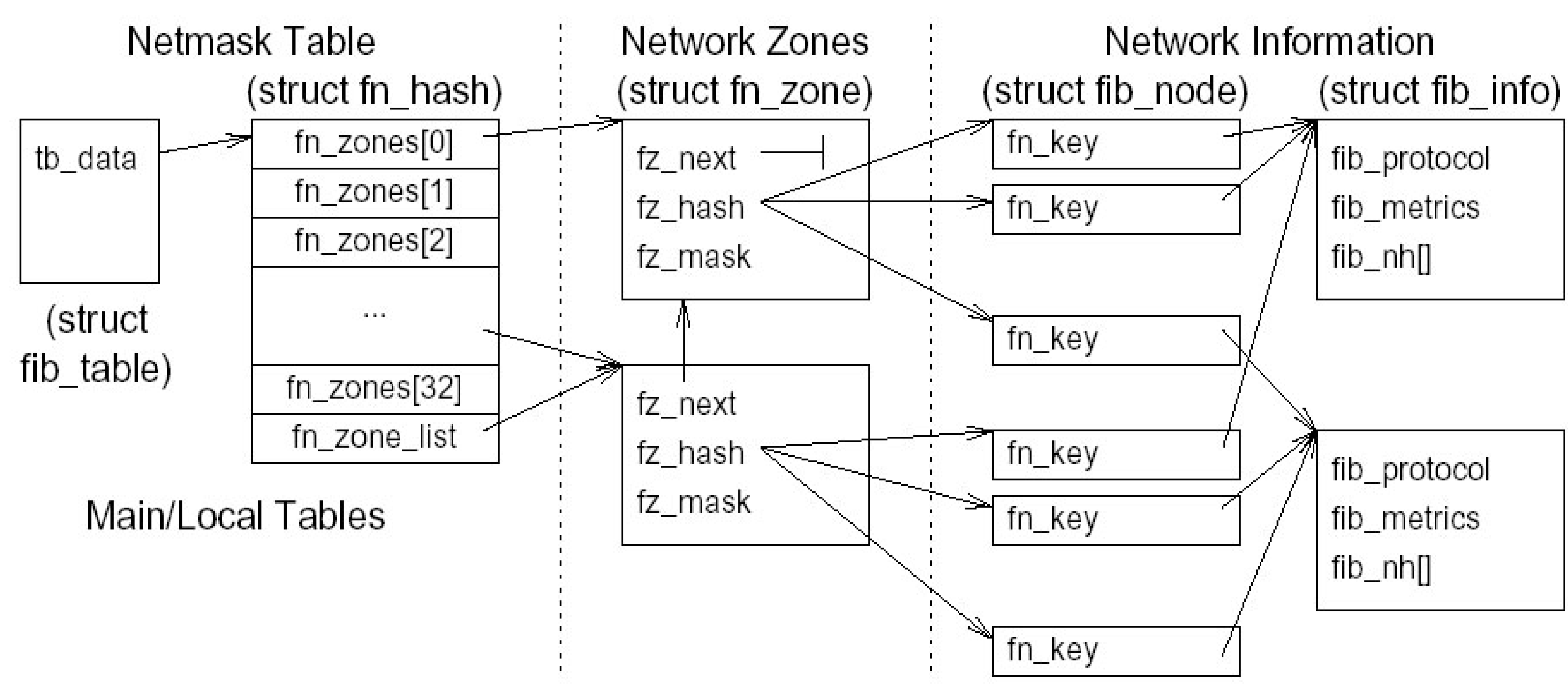
Three sets of routing data

- Directly connected
  - Neighbor Table (use ARP to maintain)
- Indirectly connected
  - FIB (Forwarding Information Base) Table
  - Routing Cache (faster)

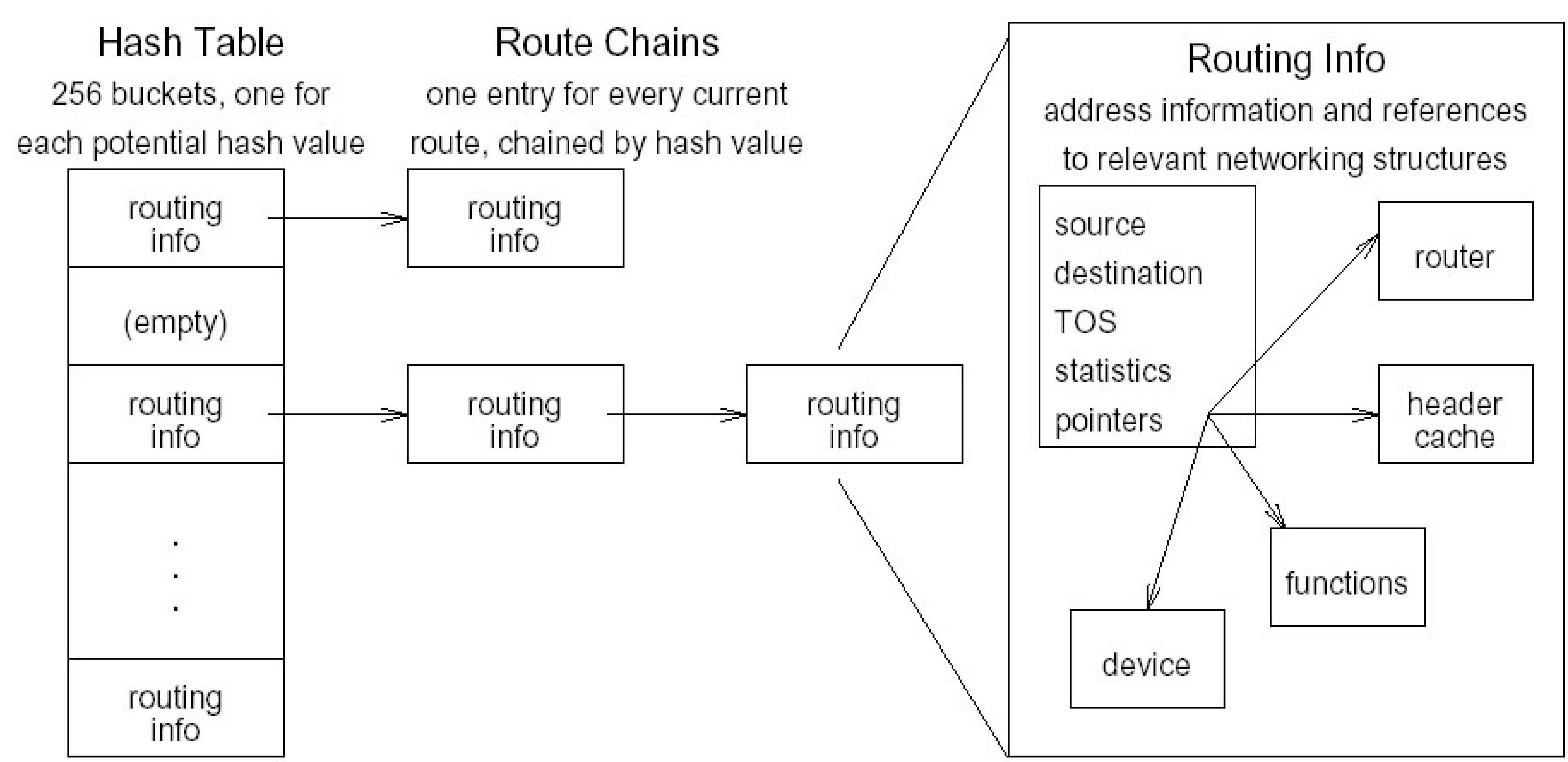
FIB conceptual organization



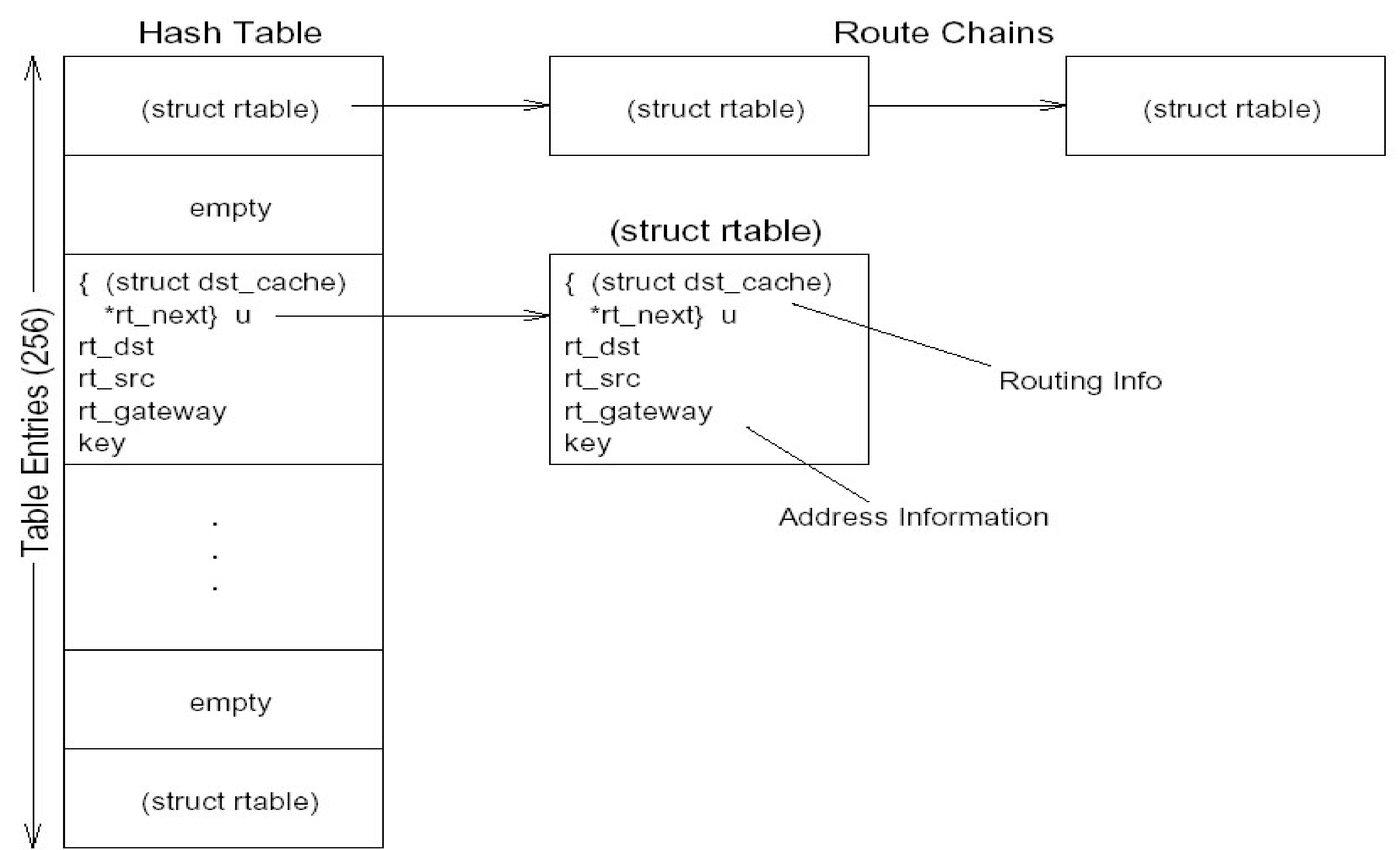
FIB data relationships



Routing Cache concept



Routing Cache data relationships



Reference

- Linux IP Stacks Commentary
  - by Stephen T. Satchell, H. B. J. Clifford
- Linux IP Networking - by Glenn Herrin
  - http://www.kernelnewbies.org/documents/
- Cross Referencing Linux
  - http://lxr.linux.no (kernel 2.4.19)