
# System Programming

Lecture #12

IP in Linux

# Contents

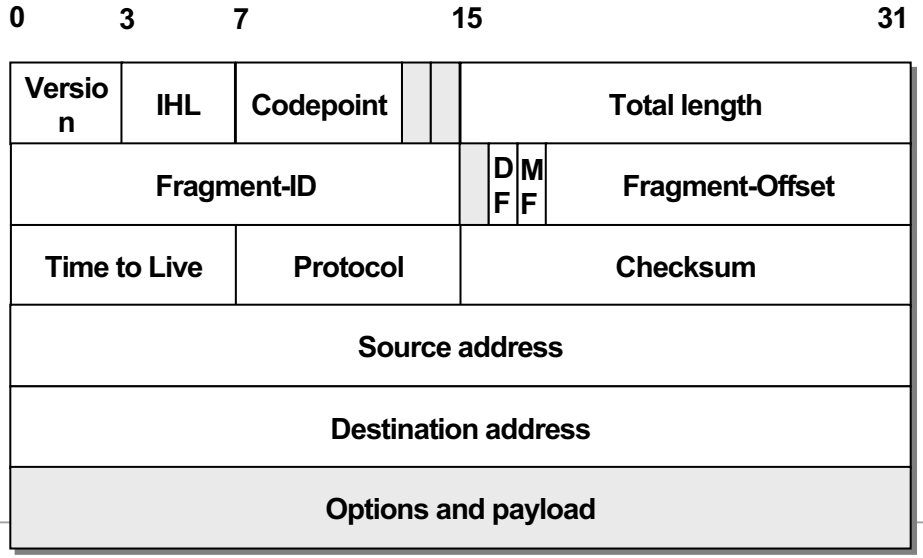
- IP (Internet Protocol)
  - IP Reviews
  - Data Structure for IP
- IP Main Function: Routing
  - IP Routing Procedure
  - Data structures for IP Routing
- Actual Implementation of IP
  - IP Implementation Architecture
  - Sources of Packets to IP layer
  - IP Output
  - IP Input
  - Packet Forwarding


# IP basics

- IP review
- Data structures for IP

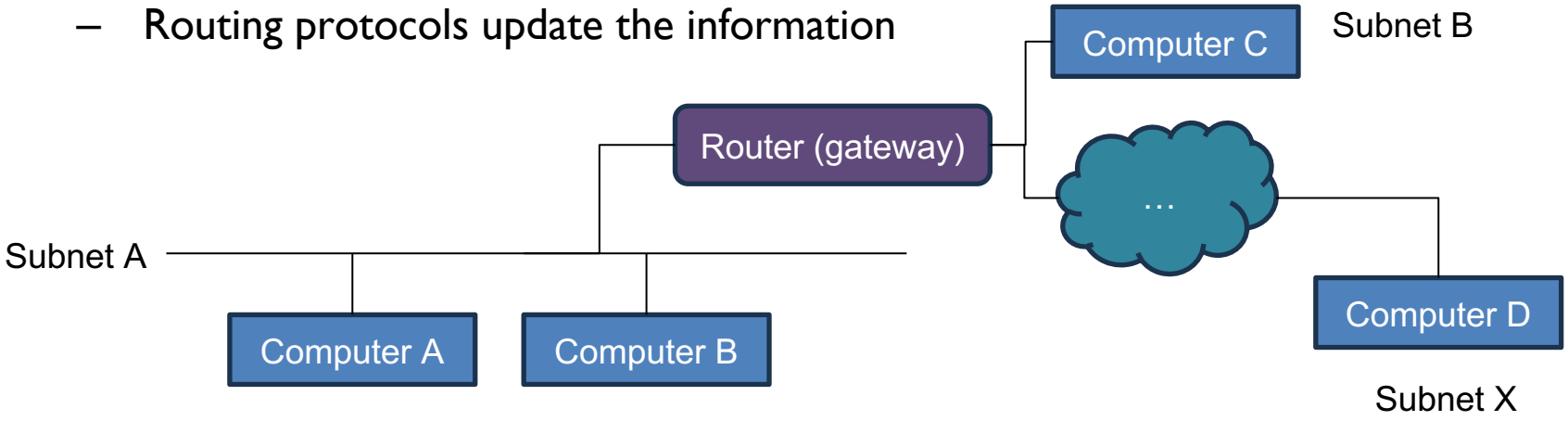
# IP Reviews

- Routing: decide next hop of datagrams toward the destination
- Fragment/reassemble datagrams
  - For the maximum transmission unit
- Best effort (unreliable)
  - If reliability is required, TCP should be used in the upper layer



# Routing

- IP deals with IP addresses (logical addresses of hosts)
  - With the IP address, the IP determines the route for a packet
- If the destination is in the local (same subnet),
  - The next hop is the destination itself
- If not (different subnet),
  - The next hop is a router (gateway) that connects to other networks
- IP manages the information for decision-making
  - Routing protocols update the information



# Routing

- Layer 2 requires physical address
  - IP layer gets the physical address for the decided next hop's IP address
- ARP (address resolution protocol)
  - Once the next hop's IP address is known
  - To get the physical address of the IP address, we use ARP
    - IP layer sends an ARP request packet with a curious IP address by broadcasting to its subnet
    - Upon receiving the ARP request, the network interface corresponding to the IP address replies to the sender with its physical address
- IP passes the physical address of the next hop to layer 2

# Data structure for IP

- `struct iphdr`

**Where** include/linux/ip.h

  - Represent the IP header of each packet

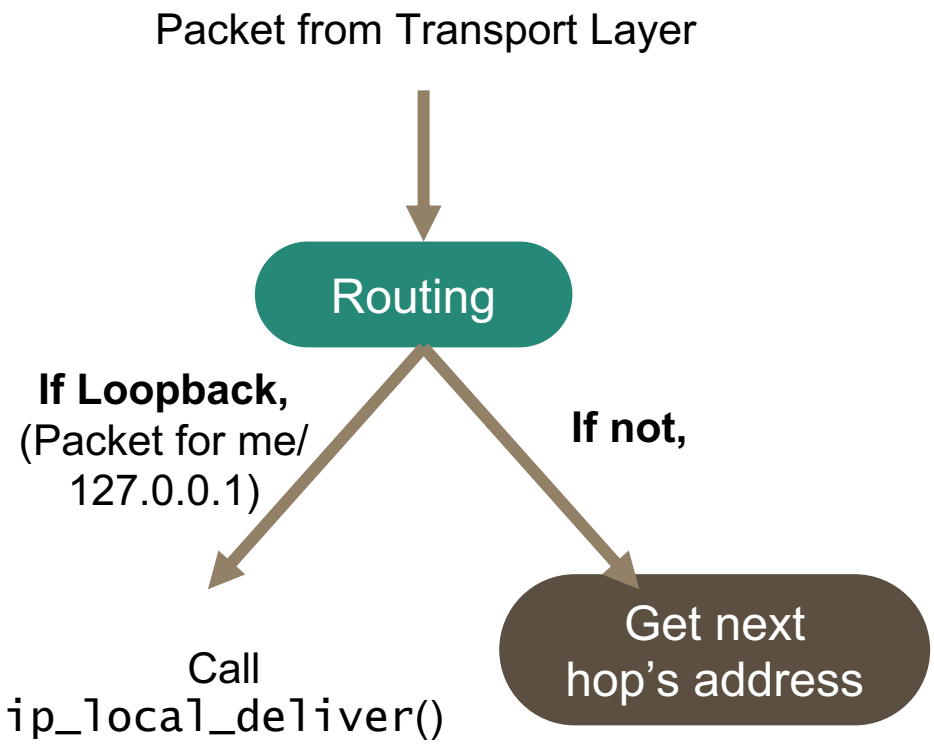
```
 85 struct iphdr {
 86 #if defined(__LITTLE_ENDIAN_BITFIELD)
 87         __u8    ihl:4,
 88                 version:4;
 89 #elif defined (__BIG_ENDIAN_BITFIELD)
 90         __u8    version:4,
 91                 ihl:4;
 92 #else
 93 #error  "Please fix <asm/byteorder.h>"
 94 #endif
 95         __u8    tos;        96         __be16  tot_len;
 97         __be16  id;         98         __be16  frag_off;
 99         __u8    ttl;       100         __u8    protocol;
101         __sum16 check;     102         __be32  saddr;
103         __be32  daddr;
104         /*The options start here. */
105 };
```


# IP implementation: Routing

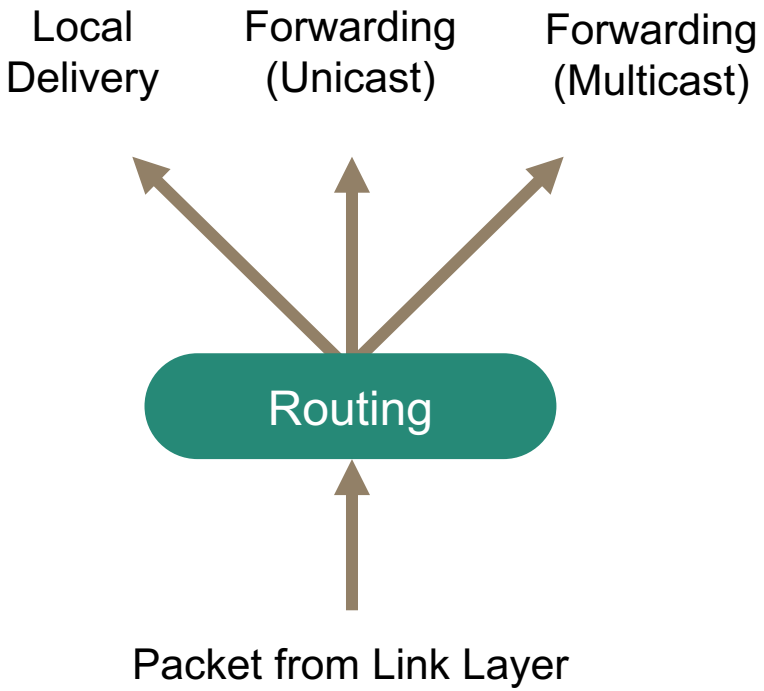
- Procedure
- Data structures for IP Routing (forwarding)
  - FIB (Forwarding Information Base)
  - Neighbor table

# IP Routing

- Sending

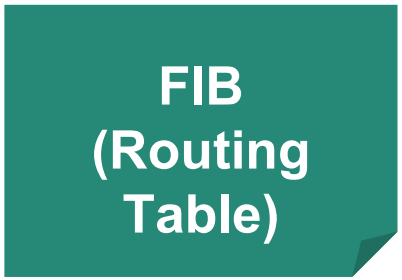



- Receiving



# Two major structures for IP Routing

- For determining next hop:
  - Forwarding information base (`struct fib_tables`)
  - Get next hop's IP address
  - Working with routing protocols
- For physical address of the next hop:
  - Neighbor Table (struct neigh_table)
  - Get physical address
  - Working with ARP

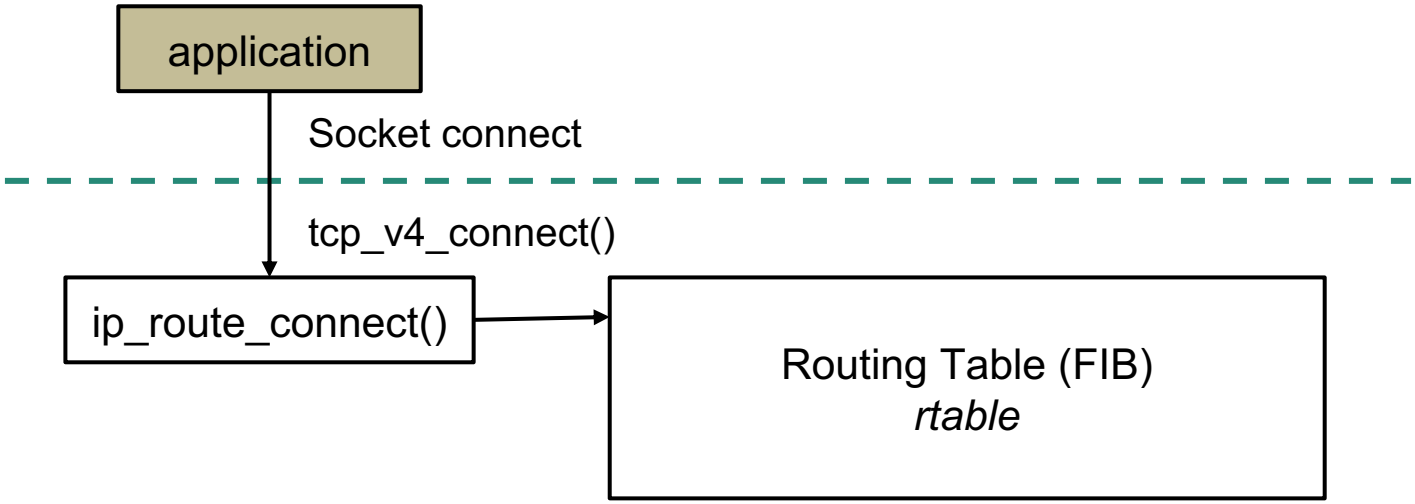




*routing cache was removed in recent kernels

# FIB

- When a user calls connect() in an application,
  - Kernel invokes `ip_route_connect()` to find a route to the destination IP address
  - Forwarding Information Base (FIB) is retrieved
    - FIB can contain multiple tables (e.g., main_table, local_table, default_table)
    - Routing lookup result determines the output device and next hop
- Result stored in `sk_dst_cache` field of `struct sock`



# FIB example

- FIB

| Prefix | Gateway | Interface |
|---|---|---|
| 192.168.1.0/24 | — (direct) | eth0 |
| 10.0.0.0/8 | 192.168.1.1 | eth0 |
| default | 192.168.1.1 | eth0 |

- If the destination IP is 10.10.10.10,
  - `fib_lookup()` → matches by the second entry
    - /8 indicates the bits to match with the destination IP (mask)
  - Decision: next hop IP of 192.168.1.1 by transmitting to eth0

# Neighbor Table

- After routing, IP finds the physical address by looking up the neighbor table
- Neighbor table
  - Store IP address and physical address (e.g., MAC address) mappings
- If the corresponding MAC address does not exist,
  get the value through ARP protocol
  - ARP request and reply to get the physical address
  - Neighbor table entry is removed when it is not used for a certain time (e.g., 60 s)
- System engineers can configure permanent entries as well

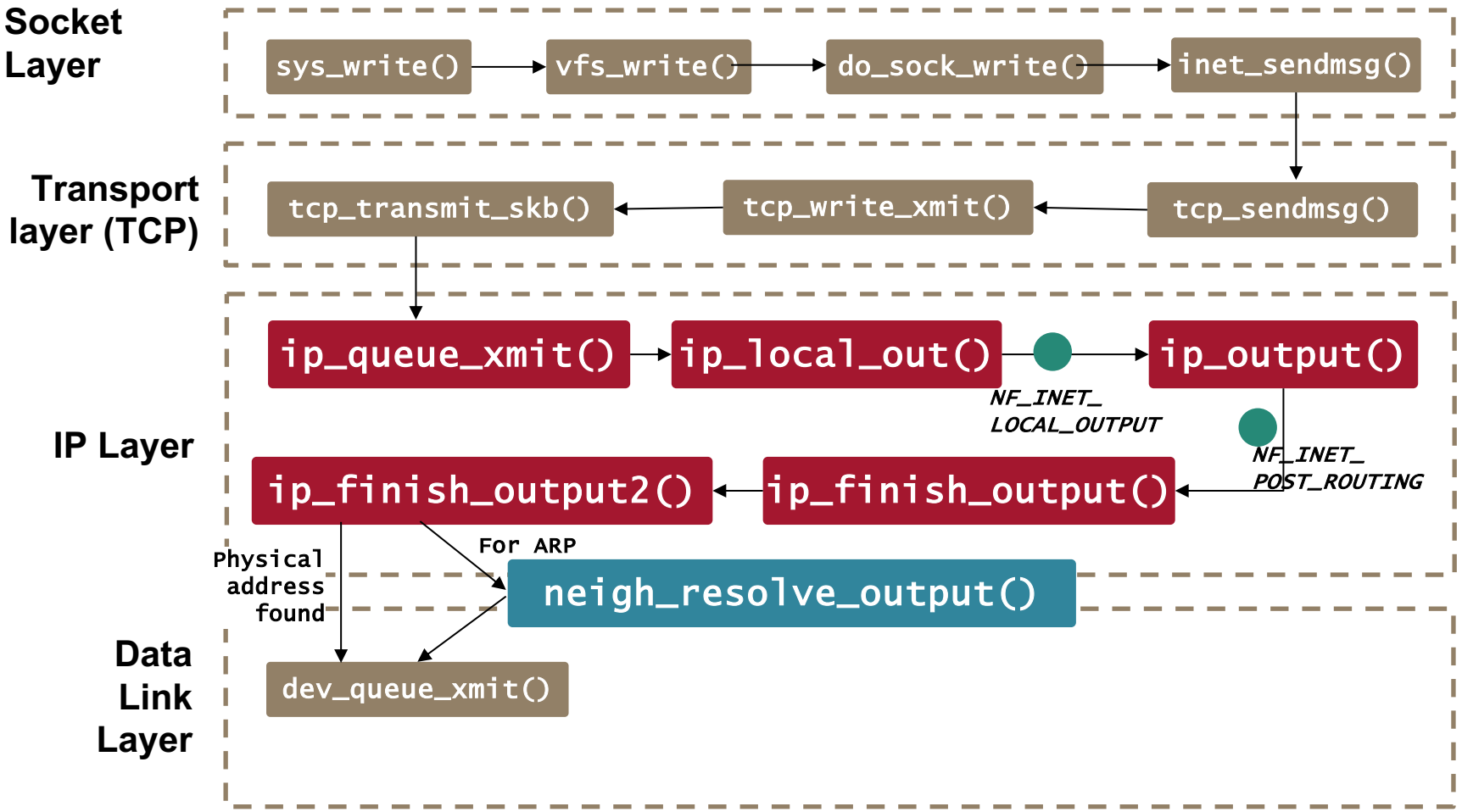

# IP implementation: TX and RX routines

- Architecture
- IP output
- IP input
- Packet forwarding

# IP layer entries

- Input (receive)
  - Packets arrive on a network interface and are passed to the ip_rcv()
- Output (transmit)
  - TCP/UDP packets are packed into an IP packet and passed down to IP via ip_queue_xmit()
- IP may internally generate packets
  - Multicast packets
  - Fragmentation of a large packet
  - ICMP/IGMP packets

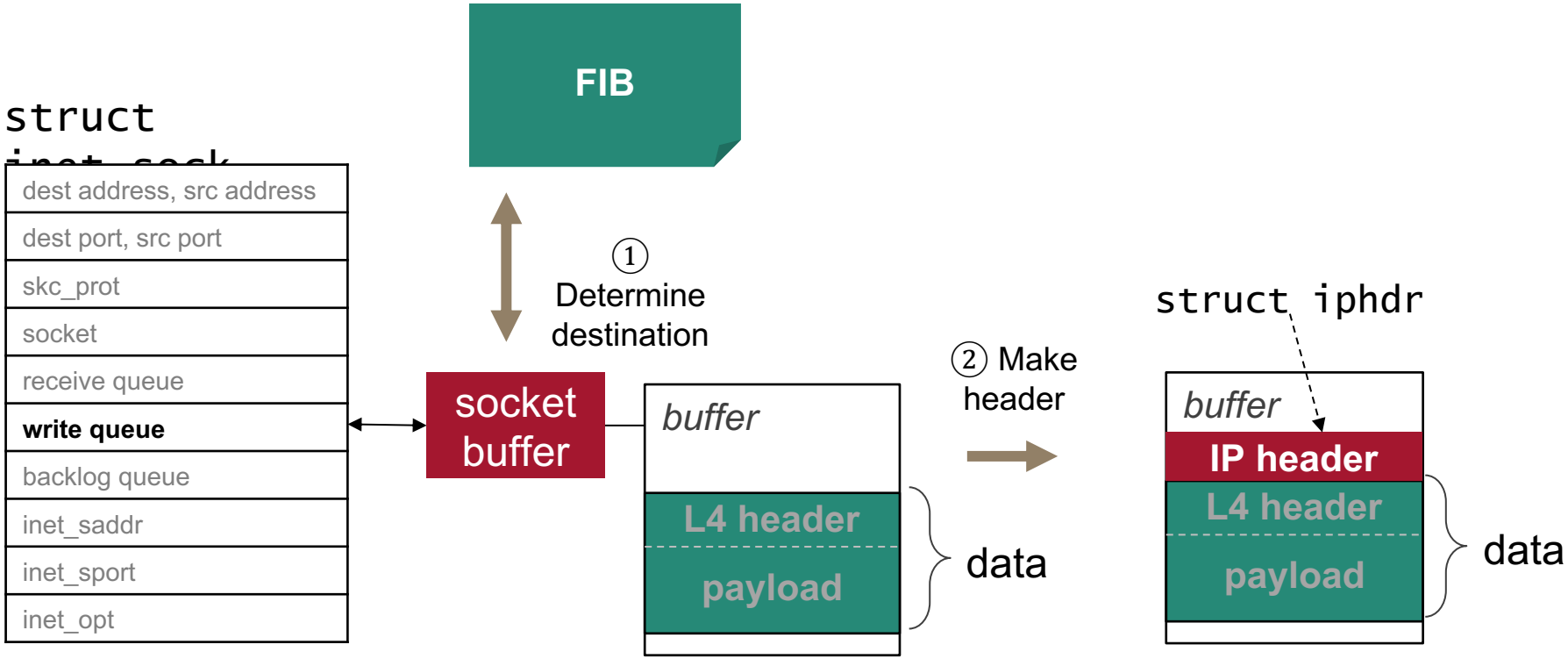
# IP output


Socket Layer
sys_write()
vfs_write()
do_sock_write()
inet_sendmsg()
Transport layer (TCP)
tcp_transmit_skb()
tcp_write_xmit()
tcp_sendmsg()
IP Layer
ip_queue_xmit()
ip_local_out()
NF_INET_ LOCAL_OUTPUT
ip_output()
NF_INET_ POST_ROUTING
ip_finish_output2()
ip_finish_output()
For ARP
Physical address found
neigh_resolve_output()
Data Link Layer
dev_queue_xmit()

# ip_queue_xmit()

- Determine destination
  - Through FIB
- Initialize IP header
  - Allocate space for the header in sk_buff
  - Initialize header: version, length, ToS, TTL, addresses, protocol, etc
- Invoke `ip_local_out()`

# ip_queue_xmit()



# ip_local_out()

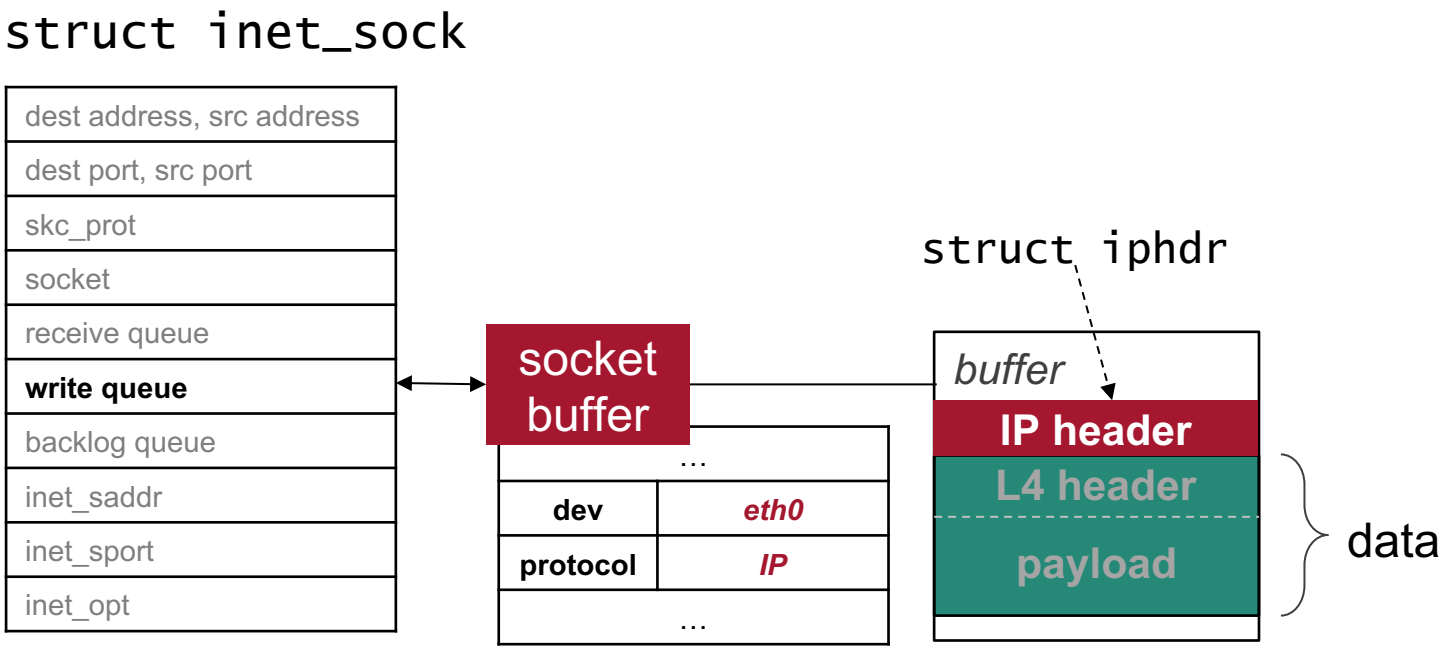
- Compute checksum
- Check whether it is for loopback or not
  - If destination is my host, call `ip_local_deliver()`
  - If not, `ip_output()` is called
- Invoke Netfilter hook (NF_INET_LOCAL_OUTPUT)
  - A tool for custom handling of packets
  - Represent a set of hooks inside the Linux kernel
  - Allow specific kernel modules to register callback functions

# ip_output()

- Update sk_buff fields
  - Specify which device to use to transmit this packet (skb->dev field)
  - Indicate that this packet uses the IP protocol (skb->protocol field)

- Invoke Netfilter hook
  - NF_INET_POST_ROUTING
  - At last, `ip_finish_output()` is called

# ip_output()


struct inet_sock
dest address, src address
dest port, src port
skc_prot
socket
receive queue
write queue
backlog queue
inet_saddr
inet_sport
inet_opt
socket buffer
…
dev
eth0
protocol
IP
…
struct iphdr
buffer
IP header
L4 header
payload
data

# ip_finish_output()

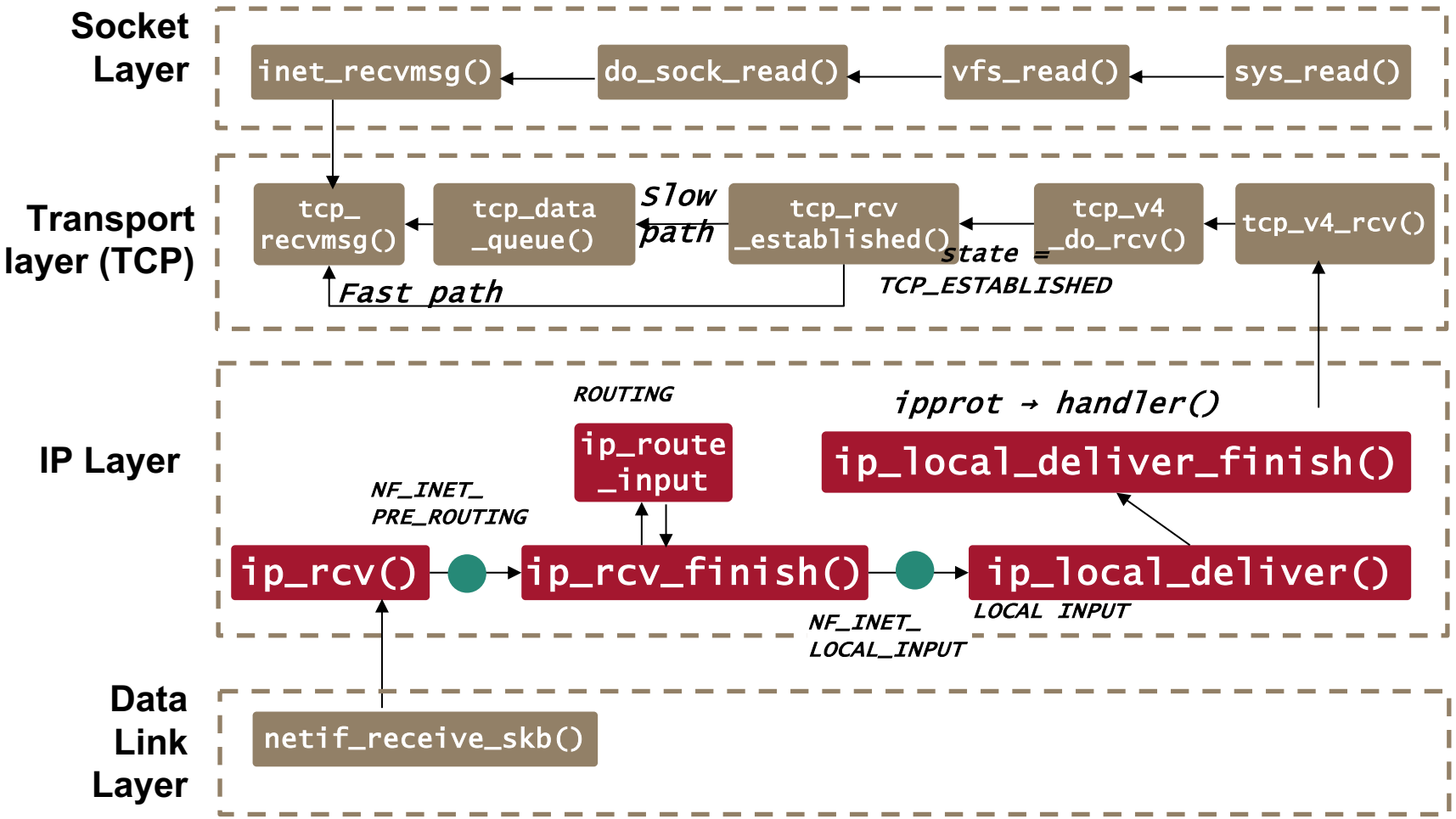
- Do fragmentation
  - Get a value for MTU (maximum transmission unit)
    - min(dev→mtu, IP_MAX_MTU)
    - E.g., IP_MAX_MTU : 65535
  - Check message length against the destination MTU

- Call either
  - If fragmentation is needed → `ip_fragment()` → each fragment is passed through `ip_finish_output()` and `ip_finish_output2()`
  - No need → `ip_finish_output2()`

# ip_finish_output2()

- Make space for L2 header
  - If skb does not have sufficient room for the L2 header, allocate L2 header's space
- Determine the physical address
  - Get a next hop's L2 address (MAC) from struct neigh_table
  - If no match is found, an ARP request is sent via `neigh_resolve_output()`
- Eventually end up in `dev_queue_xmit()`
  - Pass the packet down to the device

# IP Input


Socket Layer
inet_recvmsg()
do_sock_read()
vfs_read()
sys_read()
Transport layer (TCP)
tcp_ recvmsg()
tcp_data _queue()
Slow path
tcp_rcv _established()
tcp_v4 _do_rcv()
tcp_v4_rcv()
state = TCP_ESTABLISHED
Fast path
IP Layer
ROUTING
ip_route _input
ipprot → handler()
ip_local_deliver_finish()
NF_INET_ PRE_ROUTING
ip_rcv()
ip_rcv_finish()
ip_local_deliver()
LOCAL_INPUT
NF_INET_ LOCAL_INPUT
Data Link Layer
netif_receive_skb()

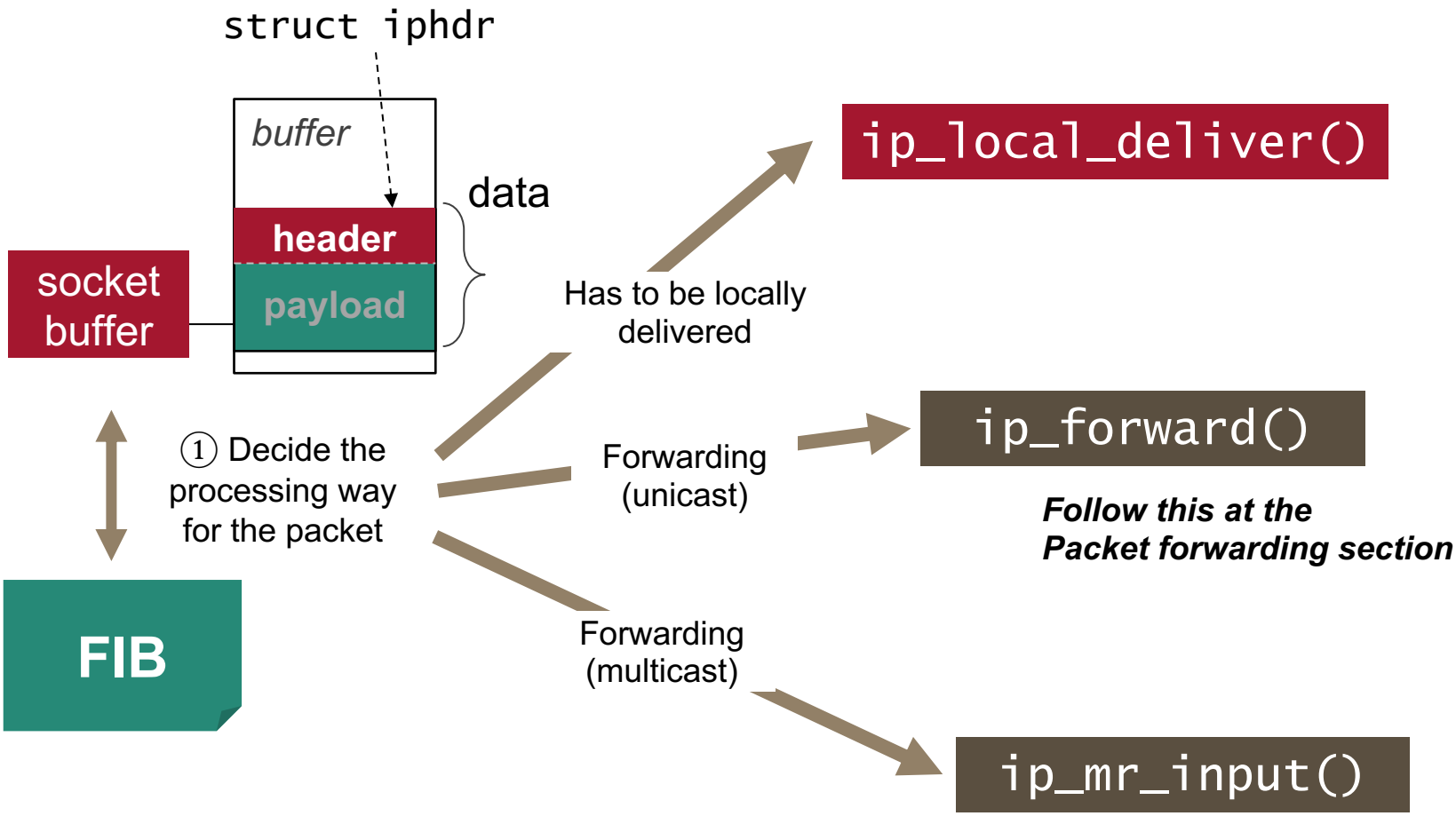
# ip_rcv()

- Do sanity checking
  - Divide region for header and data
  - Header size, IP Version, checksum, length
- Invoke Netfilter hook
  - NF_INET_PRE_ROUTING
- At last, `ip_rcv_finish()` is called

# ip_rcv_finish()

- Find the destination from FIB
- Do the action depending on the routing result
  - ip_local_deliver() – For local
  - ip_forward() – Forwarding (unicast)
  - ip_mr_input() – Forwarding (multicast)

# ip_rcv_finish()



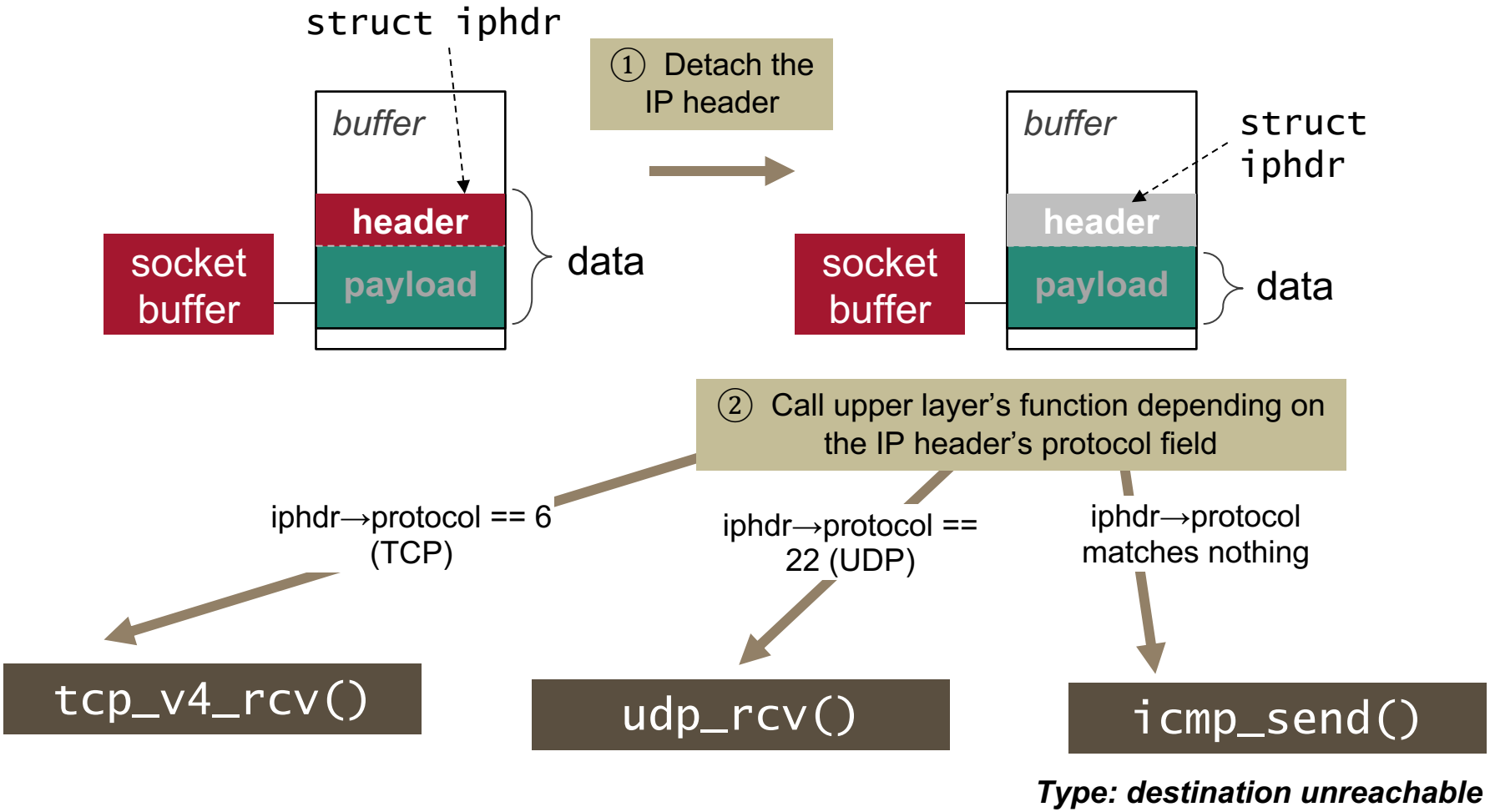
# ip_local_deliver()

- Do re-assemble
  - If the flag of the IP header indicates the packet is fragmented, the packet is reassembled
- Invoke Netfilter hook
  - NF_INET_LOCAL_IN
  - At last, ip_local_deliver_finish() is called

# ip_local_deliver_finish()

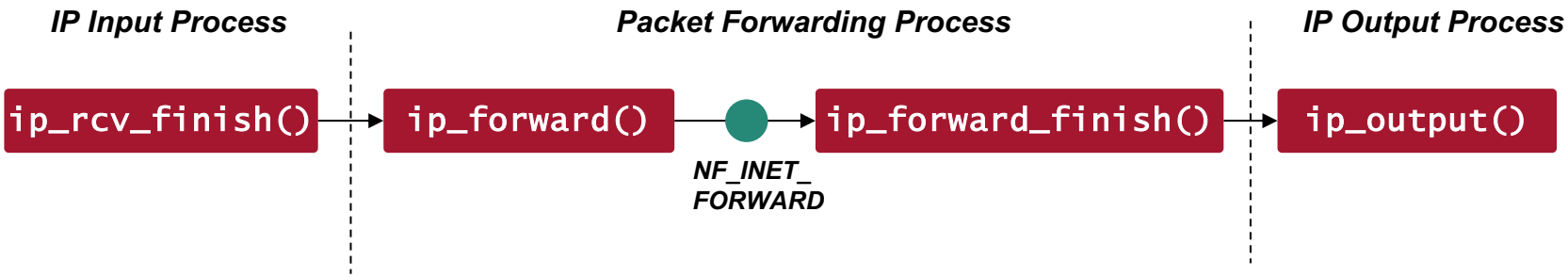
- Remove the IP header from skb
- Invoke upper layer's handler (ipprot->handler) based on the protocol field of IP header
  - `tcp_v4_rcv()`: TCP
  - `udp_rcv()`: UDP
  - `icmp_rcv()`: ICMP
  - `igmp_rcv()`: IGMP
- If no match
  - Drop it and send an ICMP message (alerting destination unreachable)

# ip_local_deliver_finish()



# Packet Forwarding

- When the packet is determined to be forwarded, ip_forward() is called



# ip_forward()

- Validate and Check
  - TTL <= 1: Drop and send ICMP (ICMP_TIME_EXCEEDED)
  - Length > MTU: Drop and send ICMP (ICMP_FRAG_NEEDED)
- Make space for L2 header
  - Check skb and ensure space is available for the L2 header
- Decrease TTL by 1
- Invoke netfilter hook
  - NF_INET_FORWARDING
  - At last, `ip_forward_finish()` is called

# ip_forward_finish()

- Handle any IP options if they exist
- Call `ip_output()`

# IP Implementation Architecture