CSC 660: Advanced OS

Netfilter

Topics

1. What is a firewall?
2. Packet Filtering with iptables
3. Netfilter Architecture
4. Packet Data Structures
5. Netfilter Data Structures
6. Writing a Netfilter extension

What is a Firewall?

- A software or hardware component that restricts network communication between two computers or networks
- In buildings, a firewall is a fireproof wall that restricts the spread of a fire
  - Network firewall prevents threats from spreading from one network to another

What is a Firewall? (2)

- A mechanism to enforce security policy
  - Choke point that traffic has to flow through
  - ACLs on a host/network level
- Policy Decisions:
  - What traffic should be allowed into network?
    - Integrity: protect integrity of internal systems
    - Availability: protection from DOS attacks
  - What traffic should be allowed out of network?
    - Confidentiality: protection from data leakage

Packet Filtering

- Forward or drop packets based on TCP/IP header information, most often:
  - IP source and destination addresses
  - Protocol (ICMP, TCP, or UDP)
  - TCP/UDP source and destination ports
  - TCP Flags, especially SYN and ACK
  - ICMP message type
- Dual-homed hosts also make decisions based on:
  - Network interface the packet arrived on
  - Network interface the packet will depart on

iptables

- Linux packet filtering system
  - iptables is user command to configure
  - netfilter is internal kernel architecture
- Features
  - Packet filtering
  - Connection tracking
  - Network Address Translation

Tables

Each table is a named array of rules.

Within a table, rules are organized into chains.

Tables:

- **Filter**
  - Packet filtering (no alterations), default table.
  - Hooks: LOCAL_IN, LOCAL_OUT, FORWARD
- **NAT**
  - Network address translation.
  - Hooks: LOCAL_OUT, PREROUTING, POSTROUTING
- **Mangle**
  - Flexible packet alterations.
  - Hooks: LOCAL_OUT, PREROUTING

iptables

iptables [-t table] cmd [matches] [target]

Commands:

- -A chain rule-spec: Append rule to chain.
- -D chain rule-spec: Delete a rule from chain
- -L chain: List all rules in chain.
- -F chain: Flush all rules from chain.
- -P chain target: Set default policy for chain.
- -N chain: Create a new chain.
- -X chain: Remove a user-defined chain.

iptables Matches

-p protocol: Specify protocol to match.

tcp, udp, icmp, etc.

-s address/mask: Source IP address to match.

-d address/mask: Dest IP address to match.

--sport: Source port (TCP/UDP) to match.

--dport: Dest port (TCP/UDP) to match.

iptables Extended Matches

-m match: Specify match module to use.

Example: limit

- Only accept 3 ICMP packets per hour.
- -m limit --limit 3/hour -p icmp -j REJECT

Example: state

- Useful stateful packet filtering.
- -m state --state NEW: match only new conns
- -m state --state ESTABLISHED: match only established connections.

iptables Targets

-j ACCEPT

Accept packet.

-j DROP

Drop packet w/o reply.

-j REJECT

Drop packet with reply.

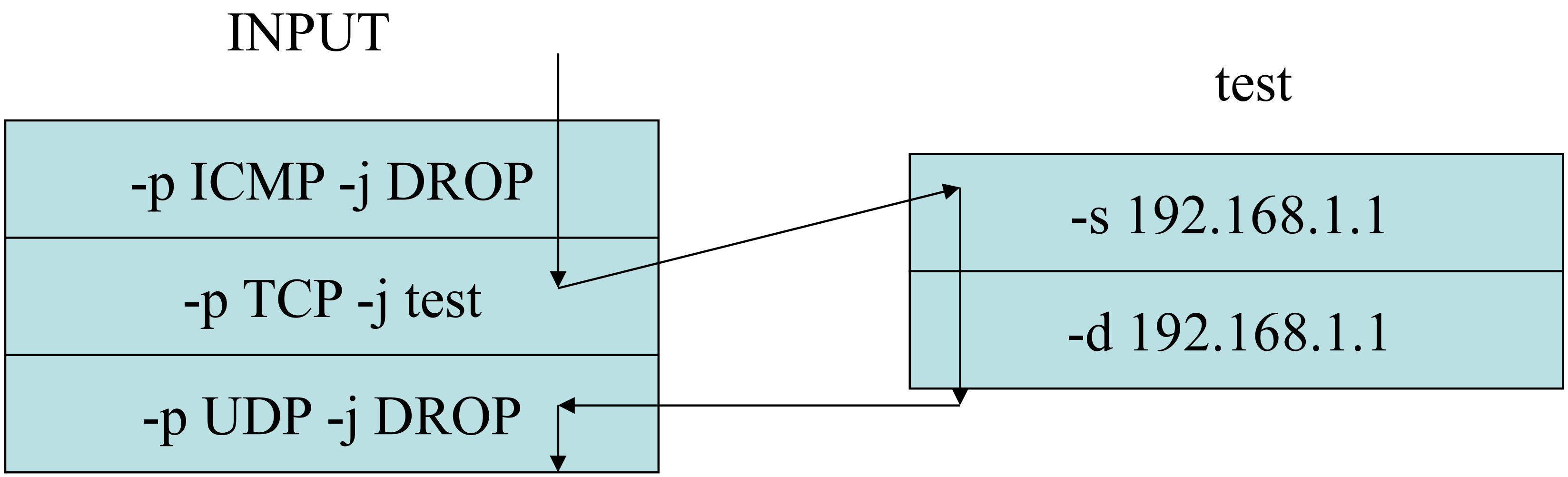
-j RETURN

Return from this chain to calling chain.

-j LOG

Log packet; chain processing continues.

Chain Targets



INPUT

-p ICMP -j DROP

-p TCP -j test

-p UDP -j DROP

test

-s 192.168.1.1

-d 192.168.1.1

Creating a Packet Filter

1. Create a security policy for a service.
   ex: allow only outgoing telnet service
2. Specify security policy in terms of which types of packets are allowed/forbidden
3. Write packet filter in terms of vendor’s filtering language

Example: outgoing telnet

- TCP-based service
- Outbound packets
  - Destination port is 23
  - Source port is random port >1023
  - Outgoing connection established by first packet with no ACK flag set
  - Following packets will have ACK flag set
- Incoming packets
  - Source port is 23, as server runs on port 23
  - Destination port is high port used for outbound packets
  - All incoming packets will have ACK flag set

Example: outgoing telnet

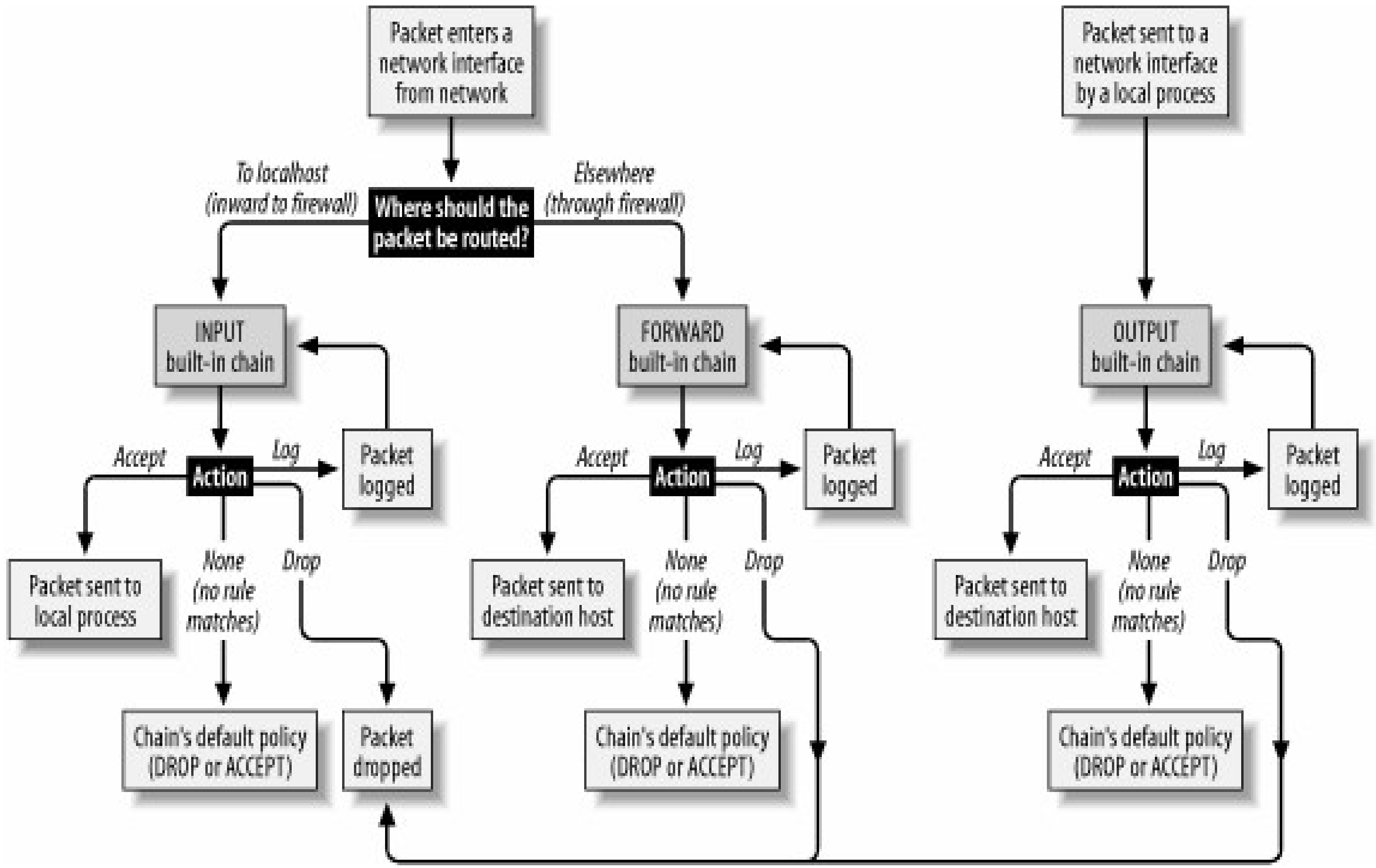
- First rule allows outgoing telnet packets
- Second rule allows response packets back in
- Third rule denies all else, following Principle of Fail-Safe Defaults

Dir	**Src**	**Dest**	**Proto**	**S.Port**	**D.Port**	**ACK?**	**Action**
Out	Int	Any	TCP	>1023	23	Either	Accept
In	Any	Int	TCP	23	>1023	Yes	Accept
Either	Any	Any	Any	Any	Any	Either	Deny

Implementing the Filter with iptables

```
# iptables –A INPUT -m state --state NEW -m
  tcp -p tcp --dport 23 -j ACCEPT
# iptables -A INPUT -m state --state
  ESTABLISHED,RELATED –m tcp –d tcp
  --sport 23 -j ACCEPT
# iptables -A INPUT -j REJECT
```

Packet Filtering Hooks


Packet enters a network interface from network
To localhost (inward to firewall)
Where should the packet be routed?
Elsewhere (through firewall)
Packet sent to a network interface by a local process
INPUT built-in chain
FORWARD built-in chain
OUTPUT built-in chain
Accept
Action
Log
Packet logged
Packet sent to local process
None (no rule matches)
Drop
Chain's default policy (DROP or ACCEPT)
Packet dropped
Accept
Action
Log
Packet logged
Packet sent to destination host
None (no rule matches)
Drop
Chain's default policy (DROP or ACCEPT)
Accept
Action
Log
Packet logged
Packet sent to destination host
None (no rule matches)
Drop
Chain's default policy (DROP or ACCEPT)

Netfilter Hooks

`LOCAL_OUT`

Hook for outgoing packets that are created locally.

`LOCAL_IN`

In `ip_local_deliver()` for incoming pkts destined for localhost.

`PRE_ROUTING`

Hook for incoming packets in `ip_rcv()` before routing.

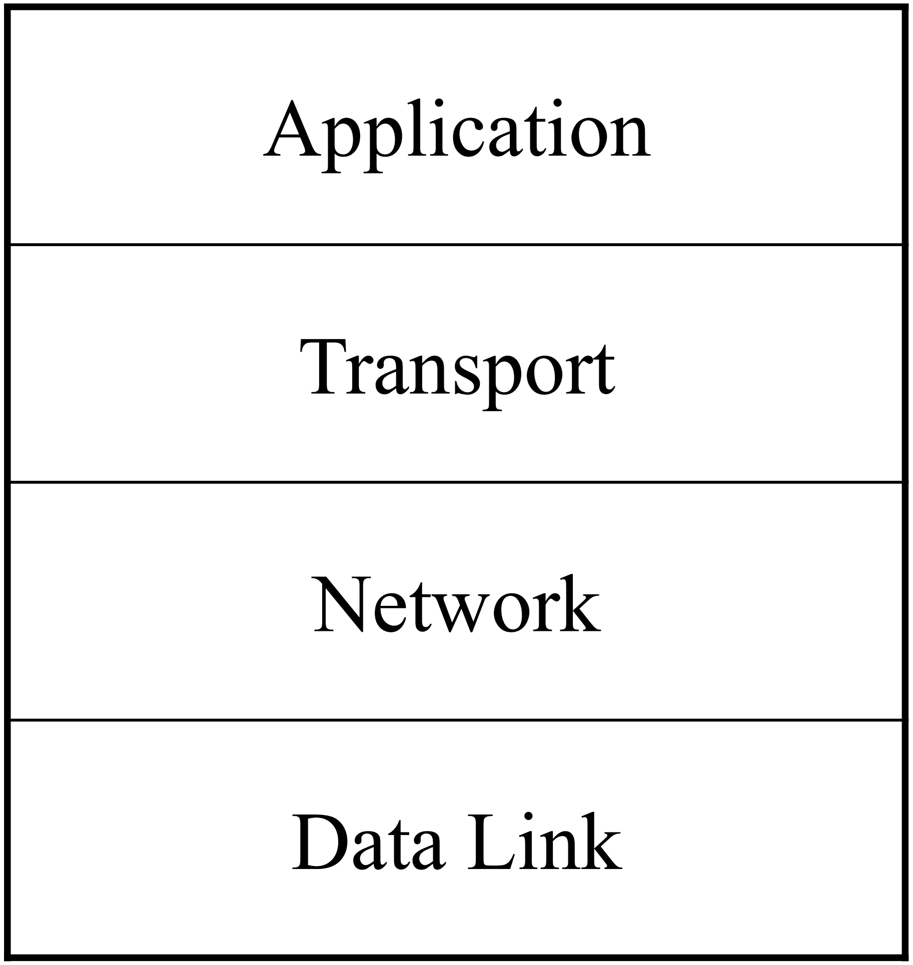
`FORWARD`

In `ip_forward()` for incoming packets destined for another host.

`POST_ROUTING`

Hook in `ip_finish_output()` for all outgoing packets.

TCP/IP Layering



Application

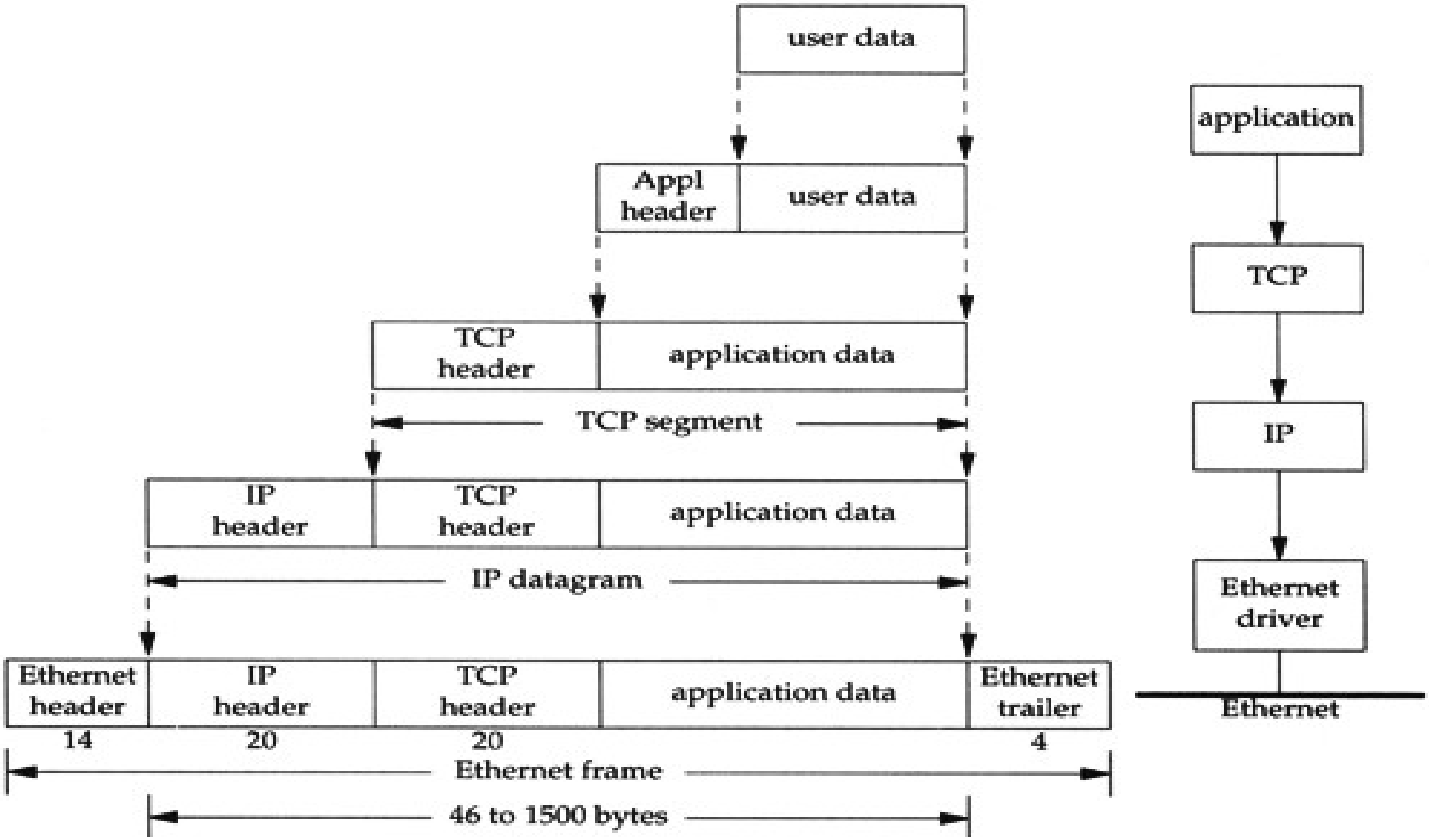
Transport

Network

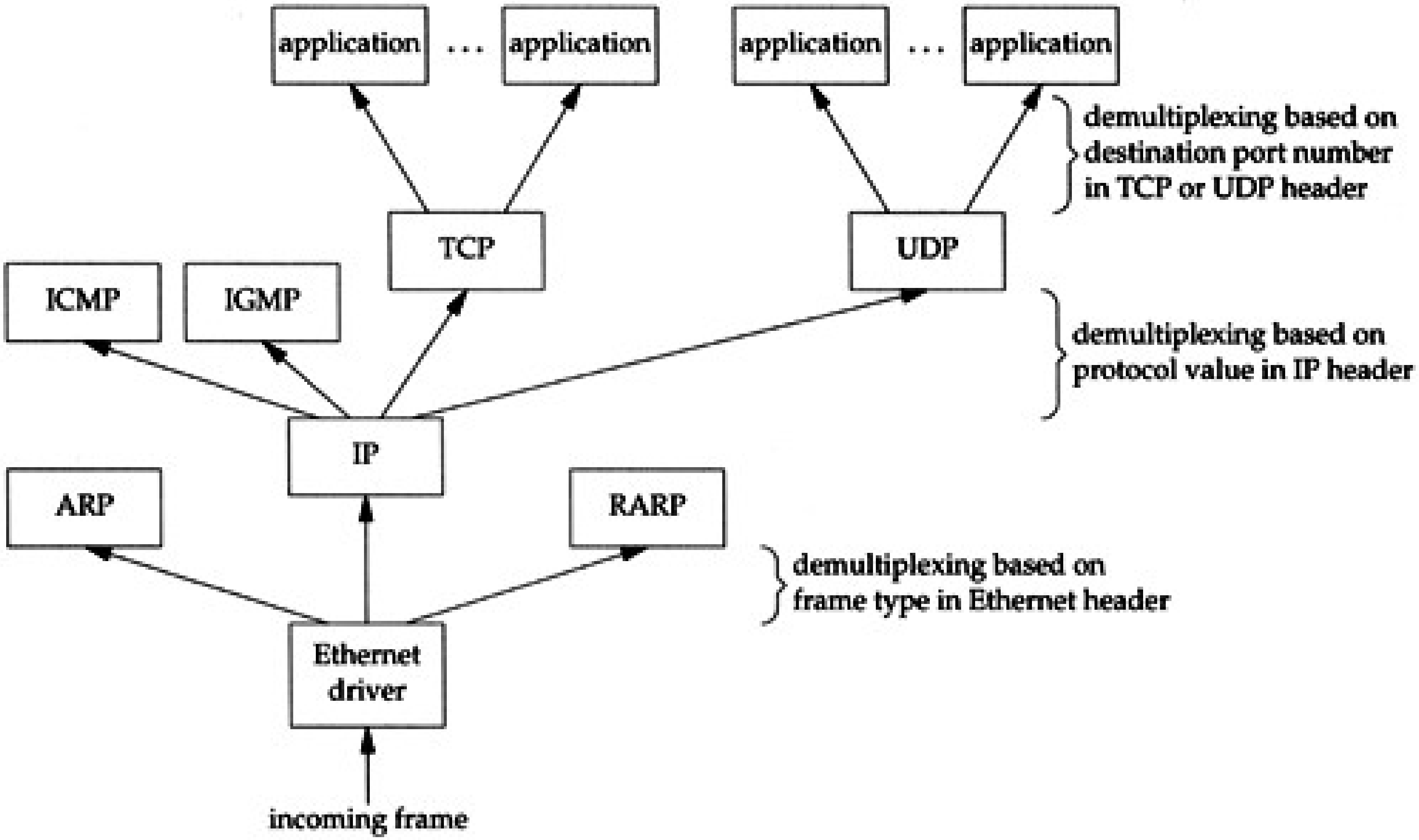
Data Link

- HTTP, FTP, telnet
- TCP, UDP
- IP, ICMP, IGMP
- Ethernet, PPP, 802.11

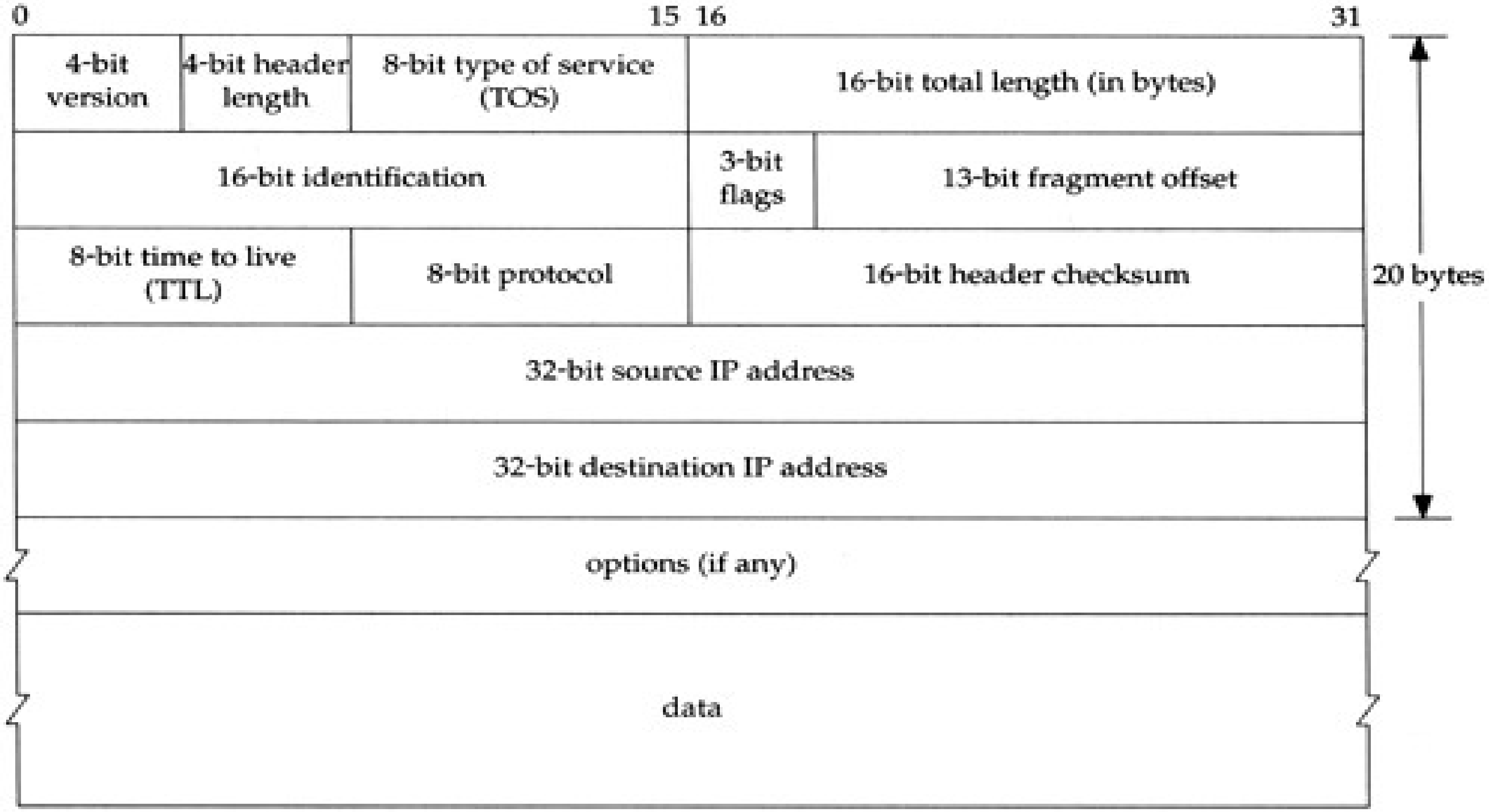
Packet Encapsulation



Packet De-multiplexing


application
...
application
application
...
application
demultiplexing based on destination port number in TCP or UDP header
TCP
UDP
ICMP
IGMP
demultiplexing based on protocol value in IP header
IP
ARP
RARP
demultiplexing based on frame type in Ethernet header
Ethernet driver
incoming frame

IP Header



0

15 16

31

4-bit version

4-bit header length

8-bit type of service (TOS)

16-bit total length (in bytes)

16-bit identification

3-bit flags

13-bit fragment offset

8-bit time to live (TTL)

8-bit protocol

16-bit header checksum

20 bytes

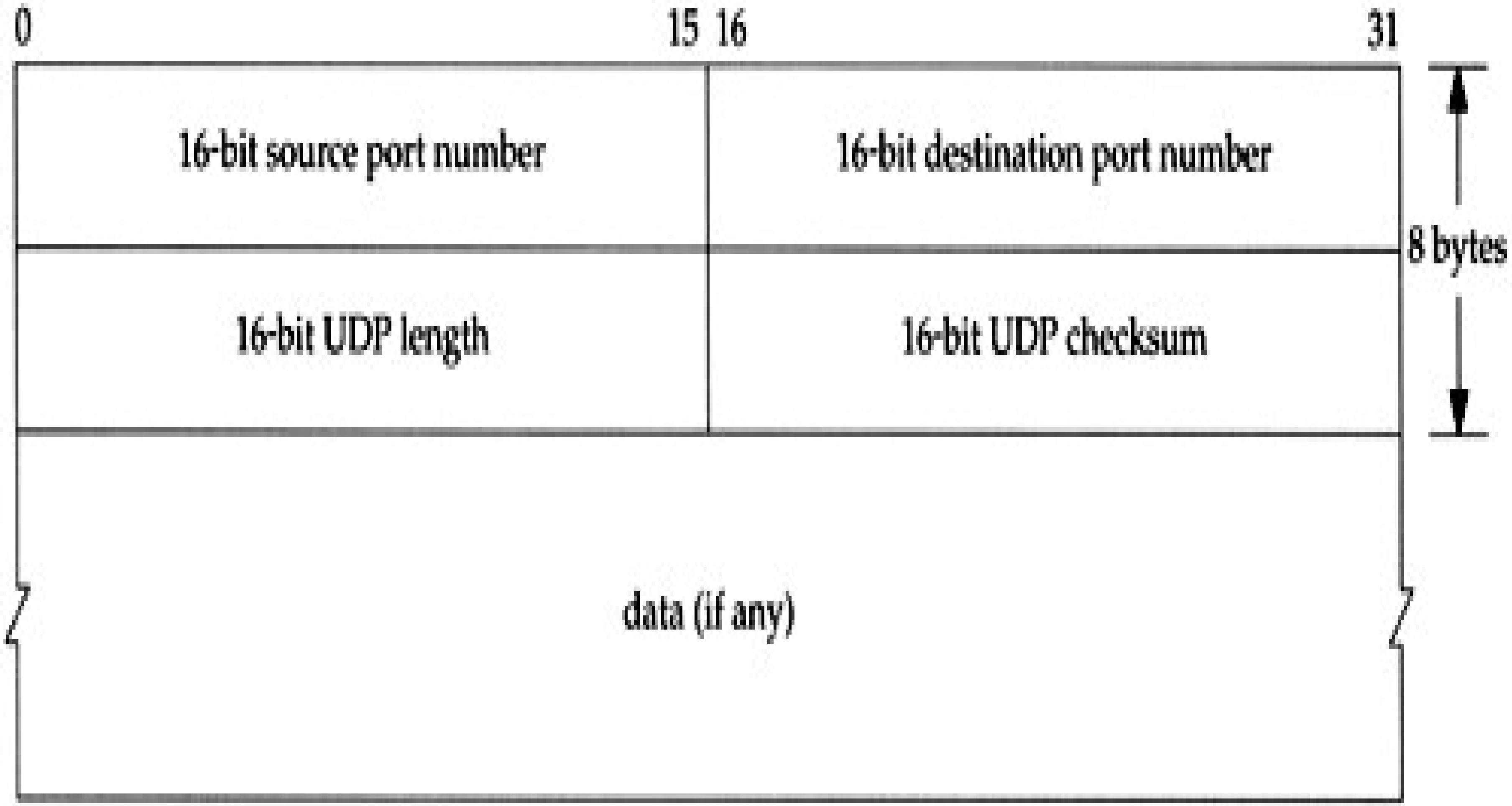
32-bit source IP address

32-bit destination IP address

options (if any)

data

UDP Header



0

15 16

31

16-bit source port number

16-bit destination port number

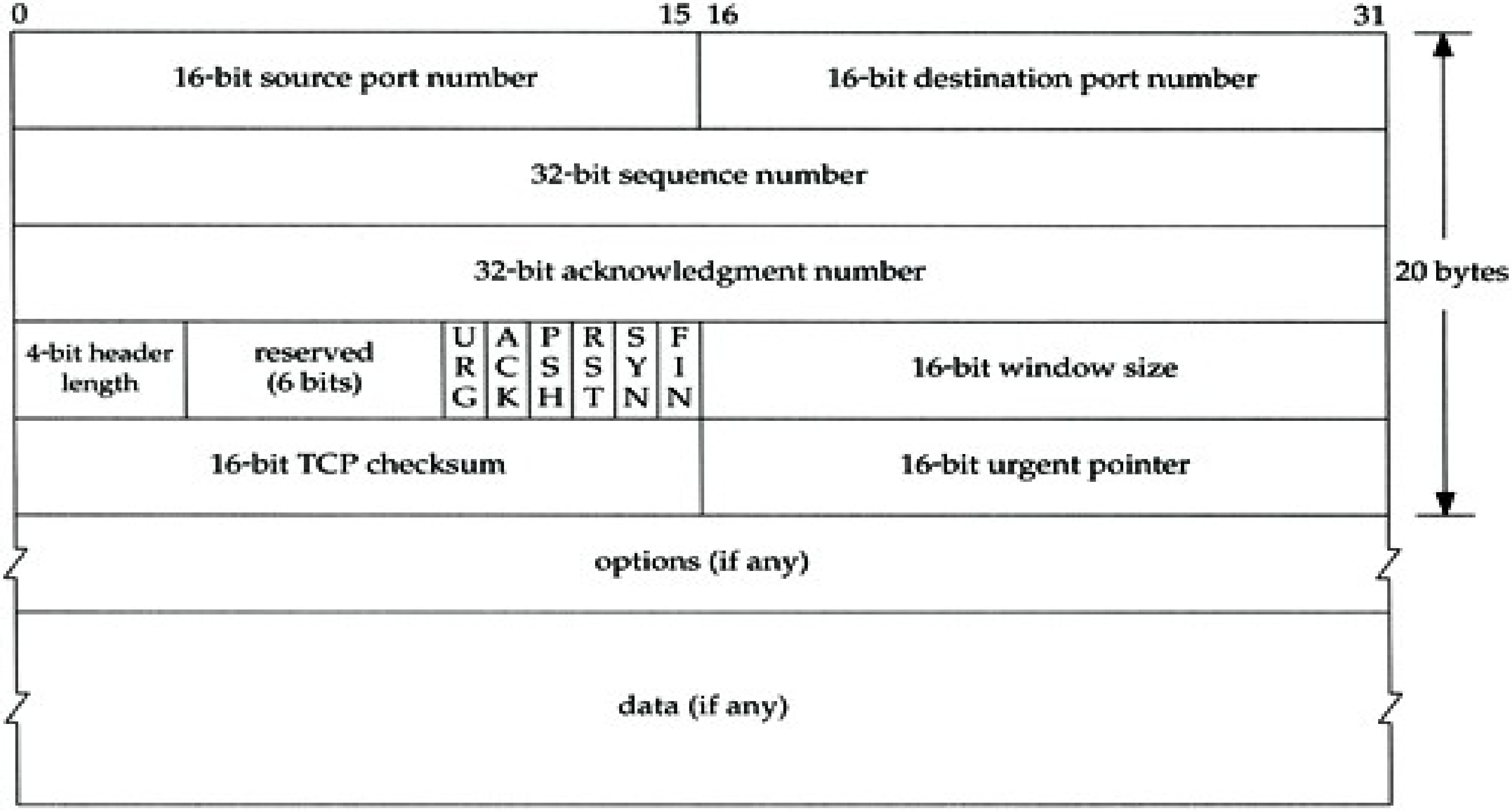
16-bit UDP length

16-bit UDP checksum

8 bytes

data (if any)

TCP Header



0 15 16 31

16-bit source port number

16-bit destination port number

32-bit sequence number

32-bit acknowledgment number

4-bit header length

reserved (6 bits)

URG ACK PSH RST SYN FIN

16-bit window size

16-bit TCP checksum

16-bit urgent pointer

options (if any)

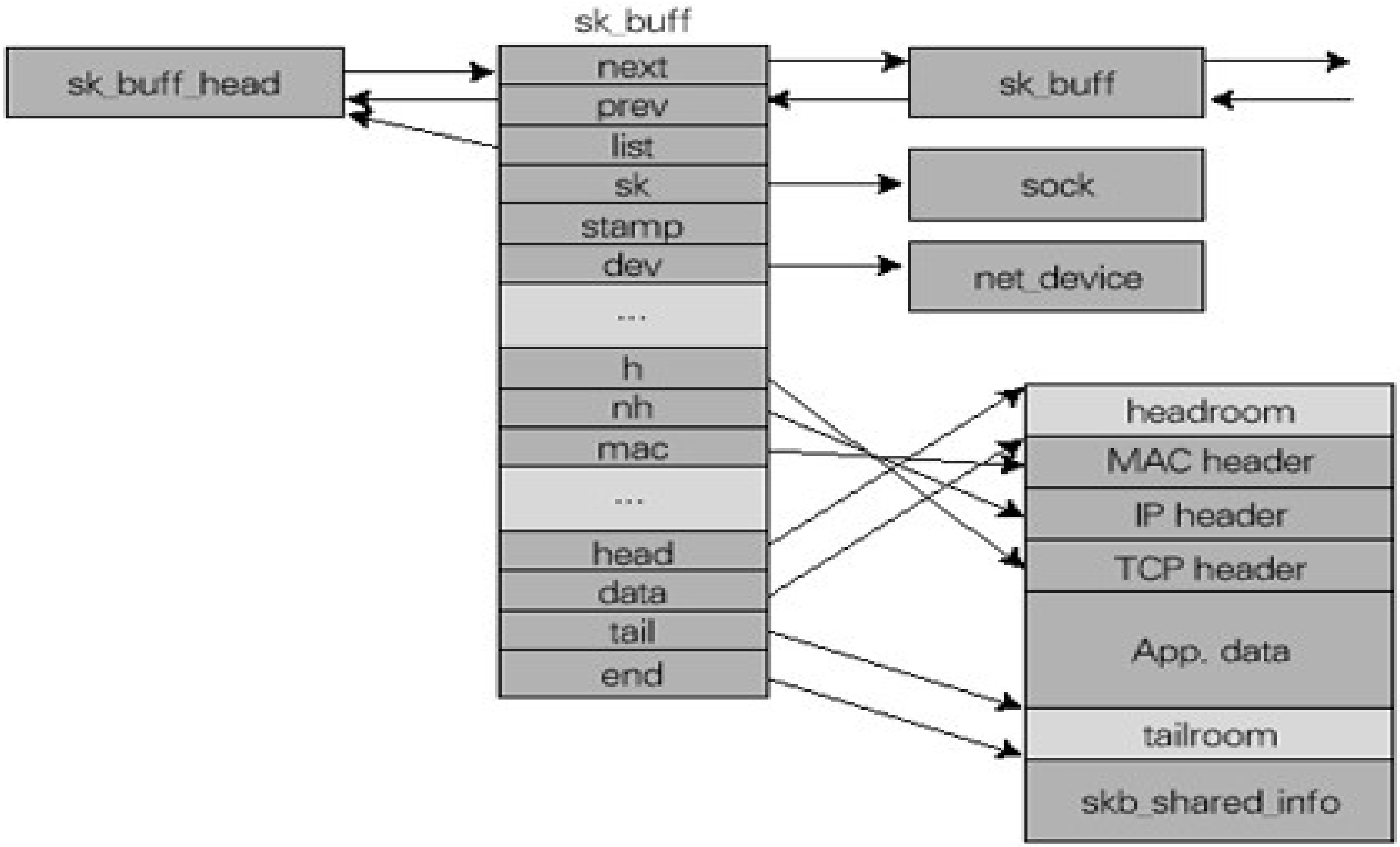
data (if any)

20 bytes

sk_buff

- Kernel buffer that stores packets.
  - Contains headers for all network layers.
- Creation
  - Application sends data to socket.
  - Packet arrives at network interface.
- Copying
  - Copied from user/kernel space.
  - Copied from kernel space to NIC.
  - Send: appends headers via skb_reserve().
  - Receive: moves ptr from header to header.

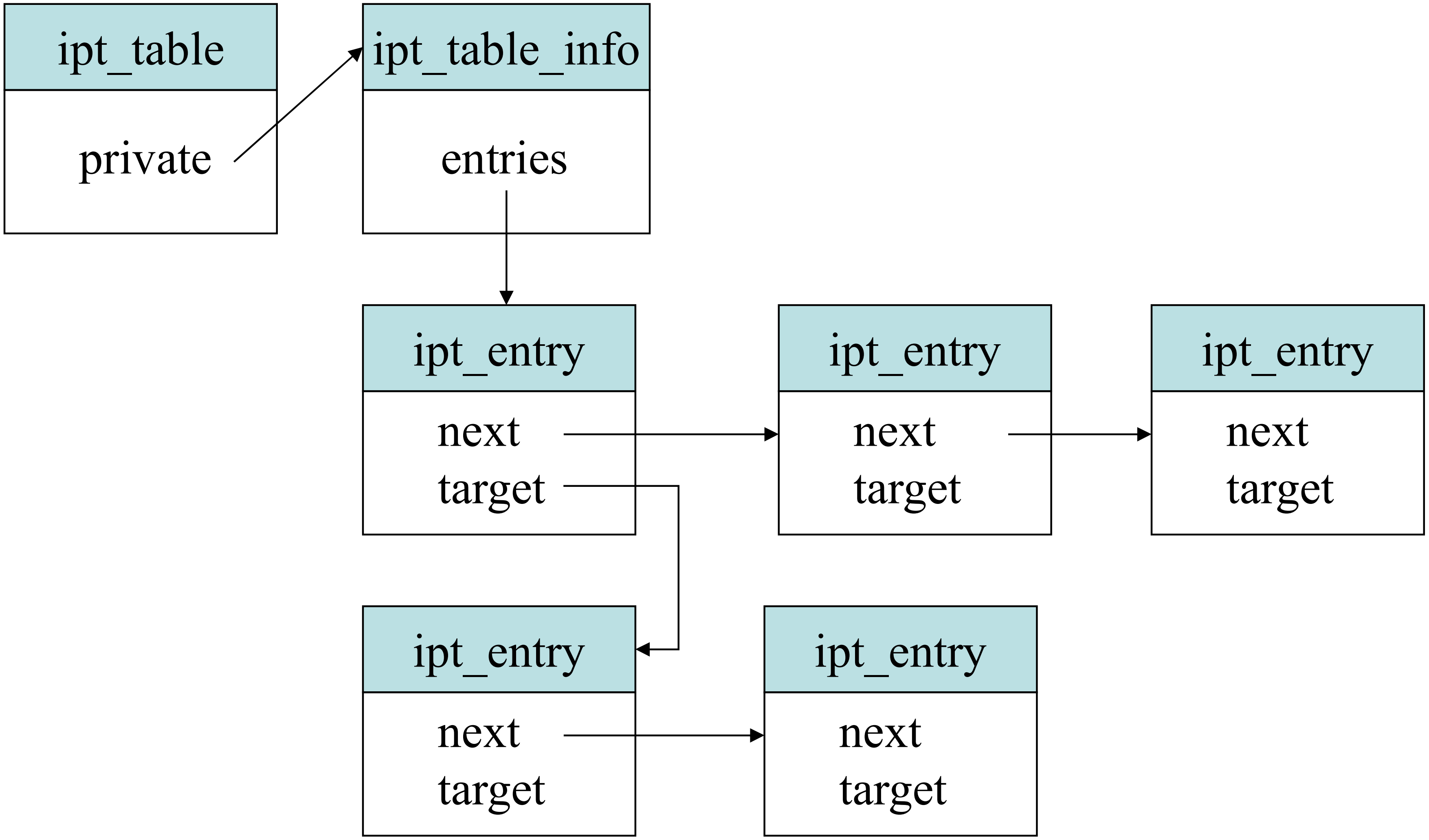
sk_buff


sk_buff
sk_buff_head
next
prev
list
sk
stamp
dev
...
h
nh
mac
...
head
data
tail
end
sk_buff
sock
net_device
headroom
MAC header
IP header
TCP header
App. data
tailroom
skb_shared_info

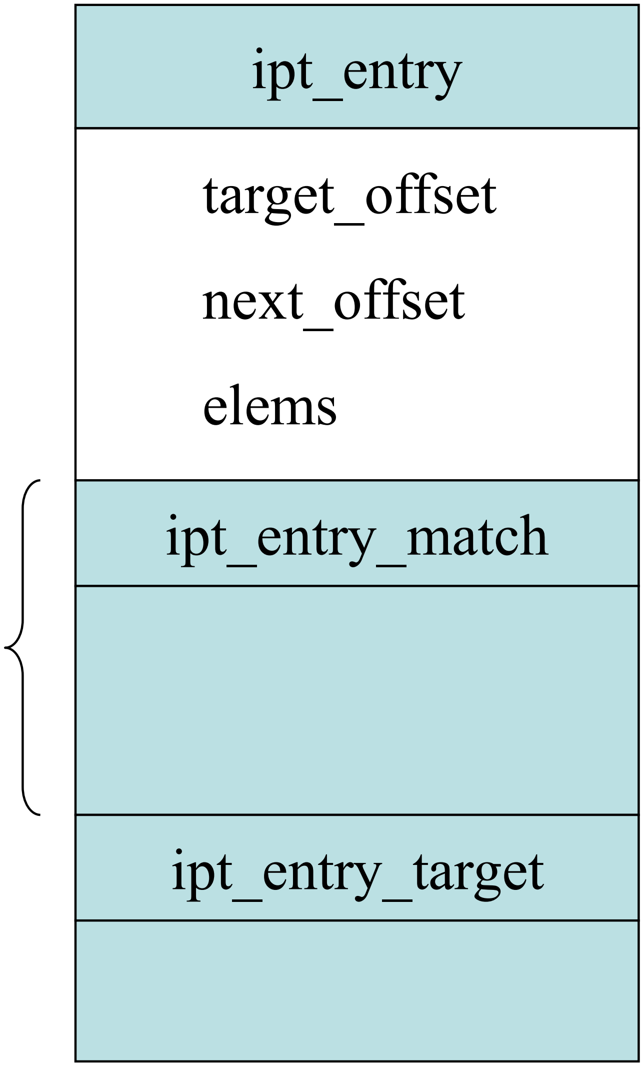
sk_buff

```
struct sk_buff {
   struct sk_buff  * next;        /* Next buffer in list  */
   struct sk_buff  * prev;        /* Previous buffer in list  */
   struct sock *sk;               /* Socket we are owned by  */
   struct timeval  stamp;         /* Time we arrived */
   struct net_device   *dev;      /* I/O net device */
   /* Transport layer header */
   union
   {
        struct tcphdr   *th;
        struct udphdr   *uh;
        struct icmphdr  *icmph;
        struct iphdr    *ipiph;
   } h;
   /* Network layer header */
   union
   {
        struct iphdr    *iph;
        struct arphdr   *arph;
   } nh;
...
};
```

IP Tables Data Structures



struct ipt_entry



ipt_entry

target_offset

next_offset

elems

ipt_entry_match

ipt_entry_target

May contain more than one match structure.

struct ipt_entry

```
struct ipt_entry
{
    /* Specifications for IP header we are to match */
    struct ipt_ip ip;
    /* Mark fields that rule examines. */
    unsigned int nfcache;

    /* Size of ipt_entry + matches */
    u_int16_t target_offset;
    /* Next ipt_entry: Size of ipt_entry + matches + target */
    u_int16_t next_offset;

    /* Back pointer */
    unsigned int comefrom;

    /* Packet and byte counters. */
    struct ipt_counters counters;

    /* The matches (if any), then the target. */
    unsigned char elems[0];
};
```

ipt_entry_match

struct ipt_entry_match contains

- Union of user and kernel structures.
  - Both contain match size.
  - Kernel part contains ptr to struct ipt_match.
- User-defined match information.

struct ipt_match

`name`: String that identifies this match.

`match`: Boolean function that determines whether the packet matched the rule or not.

`checkentry`: Boolean function that determines whether rule user attempted to enter was valid or not.

`destroy`: Called when rule deleted.

`me`: pointer to THIS_MODULE.

`ipt_entry_target`

`struct ipt_entry_target` contains

- Union of user and kernel structures.
  - Both contain target size.
  - Kernel part contains ptr to `struct ipt_target`.
- User-defined target information.

ipt_target

`name`: String that identifies target.

`target`: Returns a Netfilter action for the packet.

`checkentry`: Boolean function called when user attempts to enter this target.

`destroy`: Called when rule deleted.

`me`: pointer to THIS_MODULE.

Netfilter Actions

`NF_ACCEPT`

Allow packet to pass.

`NF_DROP`

Drop unacceptable packets

`NF_STOLEN`

Forget about packet.

`NF_QUEUE`

Queue packet for userspace program.

`NF_REPEAT`

Call this hook again.

Helper Functions

- `ipt_get_target()`
  - Returns pointer to target of a rule.
- `IPT_MATCH_ITERATE()`
  - Calls given fn for every match in given rule.
  - Function's 1st arg is struct ipt_match_entry.
  - Function returns zero for iteration to continue.
- `IPT_ALIGN()`
  - Calculates proper alignment of netfilter data structures.

`ipt_do_table`

```
Foreach ipt_entry in table:
    Foreach match in entry:
        If packet matches, call target.
            If target fn verdict is IPT_RETURN
                return to entry we were called from.
            Else if verdict IPT_CONTINUE
                continue on to next entry.
```

ipt_do_table

```
struct ipt_entry *e = get_entry(table_base,….)
unsigned int verdict = NF_DROP;
do {
     if (ip_packet_match(ip, indev, outdev, &e->ip, offset)) {
          struct ipt_entry_target *t;

          if (IPT_MATCH_ITERATE(e, do_match, …) != 0)
              goto no_match;
           t = ipt_get_target(e);  // then get verdict from target
     } else {
         no_match:
          e = (void *)e + e->next_offset;
      }
} while (!hotdrop);
if (hotdrop)
       return NF_DROP;
else return verdict;
```

do_match

```
int do_match(struct ipt_entry_match *m,
        const struct sk_buff *skb,
        const struct net_device *in,
        const struct net_device *out,
        int offset,
        const void *hdr,
        u_int16_t datalen,
        int *hotdrop)
{
   /* Stop iteration if it doesn't match */
   if (!m->u.kernel.match->match(skb, in, out, m->data,
                  offset, hdr, datalen, hotdrop))
      return 1;
   else
      return 0;
}
```

Writing a Netfilter Extension

Userspace modifications: iptables

- Place libipt_foo.c in iptables/extensions.
- Add foo to iptables/extensions/Makefile.

Kernel modifications: netfilter

- Add ipt_foo.c in net/ipv4/netfilter.
- Add ipt_foo.h in include/linux/netfilter_ipv4.

Modifying iptables: libipt_foo.c

```
static struct iptables_match sctp = {
    .name          = "sctp",
    .version       = IPTABLES_VERSION,
    .size          = IPT_ALIGN(sizeof(struct
  ipt_sctp_info)),
    .userspacesize = IPT_ALIGN(sizeof(struct
  ipt_sctp_info)),
    .help          = &help,
    .init          = &init,
    .parse         = &parse,
    .final_check   = &final_check,
    .print         = &print,
    .save          = &save,
    .extra_opts    = opts
};
void _init(void)  {   register_match(&sctp); }
```

Modifying iptables: libipt_foo.c

Functions in iptables_match handle options:

- help: iptables –h
- print: iptables –L
- parse: iptables –[A|I|D|R]
- save: iptables-save

Parsing sets `struct ipt_foo_info`

- Via (*match)->data field.
- Info struct defined in kernel include file.

Modifying Netfilter: ipt_foo.c

```
static struct ipt_match ipt_my_reg = {
  .list = { NULL, NULL }
  .name = “limit",
  .match = &match,
  .checkentry = &checkentry,
  .destroy = &destroy,
  .me = THIS_MODULE
};
static int __init limit_init(void)
{
        if (ipt_register_match(&ipt_my_reg))
                return -EINVAL;
        return 0;
}
static void __exit limit_fini(void)
{  ipt_unregister_match(&ipt_my_reg);  }
```

References

1. Michael D. Bauer, *Linux Server Security, 2nd edition*, O'Reilly, 2005.
2. Christian Benvenuti, *Understanding Linux Network Internals*, O'Reilly, 2006.
3. Daniel P. Bovet and Marco Cesati, *Understanding the Linux Kernel, 3rd edition*, O'Reilly, 2005.
4. Thomas F. Herbert, *The Linux TCP/IP Stack*, Charles River Media, 2005.
5. Jennifer Hou, "Inside Netfilter," http://lion.cs.uiuc.edu/courses/cs498hou_spring05/lectures.html, 2005.
6. Lu-chuan Kung, "Netfilter Tutorial," http://lion.cs.uiuc.edu/courses/cs498hou_spring05/lectures.html, 2005.
7. Robert Love, *Linux Kernel Development, 2nd edition*, Prentice-Hall, 2005.
8. Rusty Russell, Linux World 2000 Netfilter Tutorial, http://www.netfilter.org/documentation/tutorials/lw-2000/tut.html, 2000.
9. Rusty Russell and Harald Welte, Linux netfiler Hacking HOWTO, http://www.netfilter.org/documentation/HOWTO/netfilter-hacking-HOWTO.html, 2002.
10. Rusty Russel, Linux Packet Filtering HOWTO, http://www.netfilter.org/documentation/HOWTO//packet-filtering-HOWTO.html, 2002.