Day 4
Networking

Joel Isaacson
Ascender Technologies Ltd.
http://ascender.com

Today's Lectures

- Today we will explore Linux Networking
  - BSD sockets
  - Unix domain sockets
  - SCTP
  - TCP vs. UDP vs. SCTP
  - TCP buffering
  - Network device drivers
  - Skbuffs – Linux networking buffers

What is a socket?

- I thought everything was a file.
- They are not files but are accessed via file descriptors.
- Sockets have been introduced in BSD Unix in 1981.
- They have a very different interface than files.

BSD Sockets

- You can do normal `read/write`'s with sockets.
- Normally some variants of recv/send are used with sockets.
- They can be used with network communications between computers or within one computer.

Three Types of Data Sockets

- There are three major types of internet data sockets, corresponding to the three major transport protocols:
  - TCP/IP
  - UDP/IP
  - SCTP/IP

TCP/IP

- The Transmission Control Protocol (TCP) is one of the core protocols of the Internet Protocol Suite.
- TCP is so central that the entire suite is often referred to as "TCP/IP."
- Whereas IP handles lower-level transmissions TCP deal with end to end connectivity.

TCP/IP

- In particular, TCP provides reliable, ordered delivery of a stream of bytes from one program on one computer to another program on another computer.
- There are no record boundaries in the data stream.

TCP/IP

- Due to network congestion, traffic load balancing, or other unpredictable network behavior, IP packets can be lost or delivered out of order.
- TCP detects these problems, requests retransmission of lost packets, rearranges out-of-order packets, and helps minimize network congestion.

TCP/IP

- Once TCP at the receiving end has finally reassembled a perfect copy of the large data chunk originally transmitted, it passes that single chunk up to the application program at the receiving end.
- TCP greatly simplifies the application programmer's network communication task.

UDP/IP

- Using UDP, programs on networked computers can send short messages sometimes known as datagrams (using Datagram Sockets) to one another.
- UDP is sometimes called the Universal Datagram Protocol.

UDP/IP

- UDP does not guarantee reliability or ordering in the way that TCP does.
- Datagrams may arrive out of order, appear duplicated, or go missing notice.
- Avoiding the overhead of checking whether every packet actually arrived may make UDP faster and more efficient, for some applications.

UDP/IP

- Time-sensitive applications often use UDP because dropped packets are preferable to delayed packets.
- UDP's stateless nature is also useful for servers that answer small queries from huge numbers of clients.
- Unlike TCP, UDP is compatible with packet broadcast and multicasting.

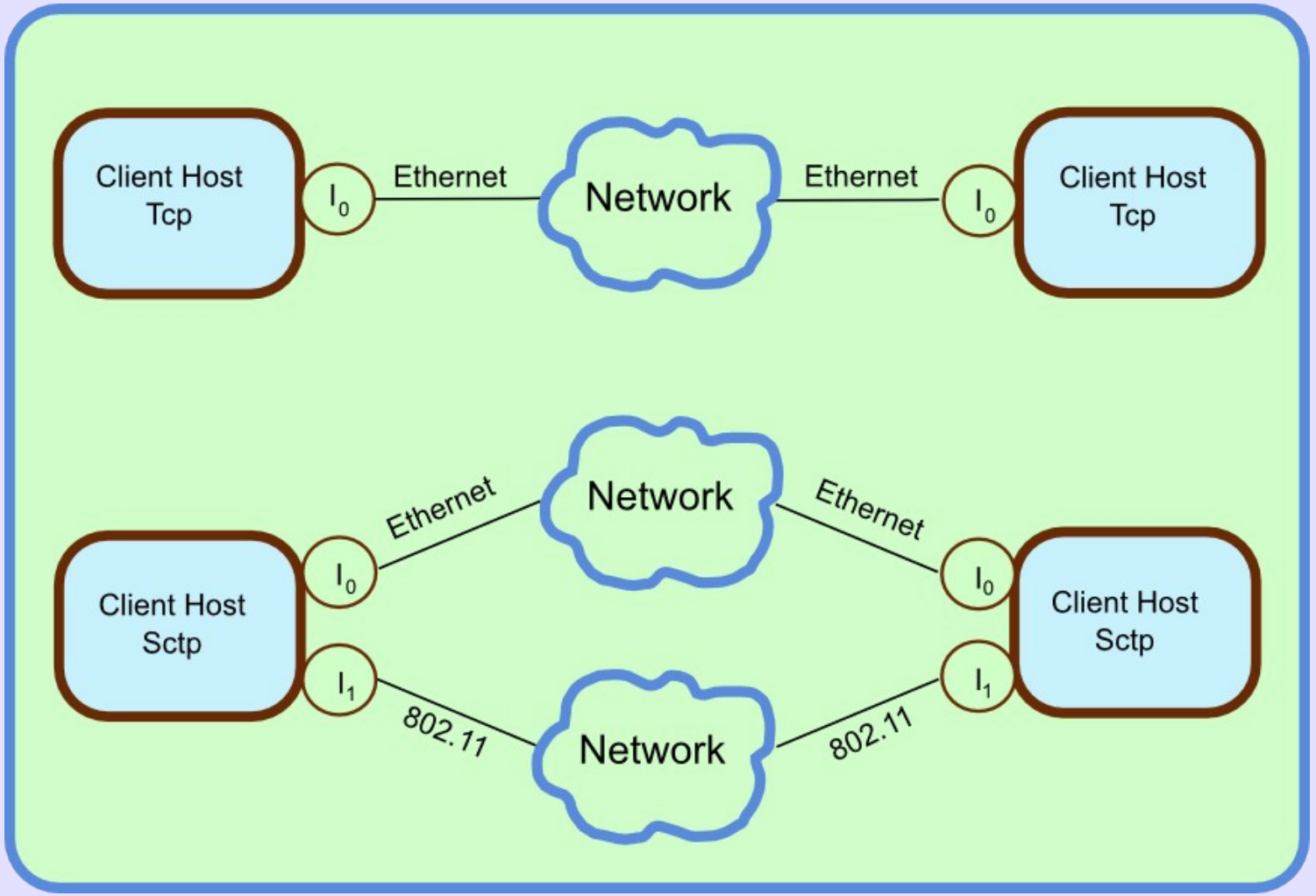
STCP/IP

- The Stream Control Transmission Protocol (SCTP) is a reliable transport protocol that provides stable, ordered delivery of data between two endpoints (much like TCP) and also preserves data message boundaries (like UDP).
- However, unlike TCP and UDP, SCTP offers such advantages as multi-homing and multi-streaming capabilities.

STCP/IP

- While the protocol was originally designed for telephony signaling (under the RFC 2960), SCTP provided added bonuses
  - Solves some of the limitations of TCP while borrowing beneficial features of UDP.
  - high availability - multi-homing
  - increased reliability
  - improved security for socket initiation.

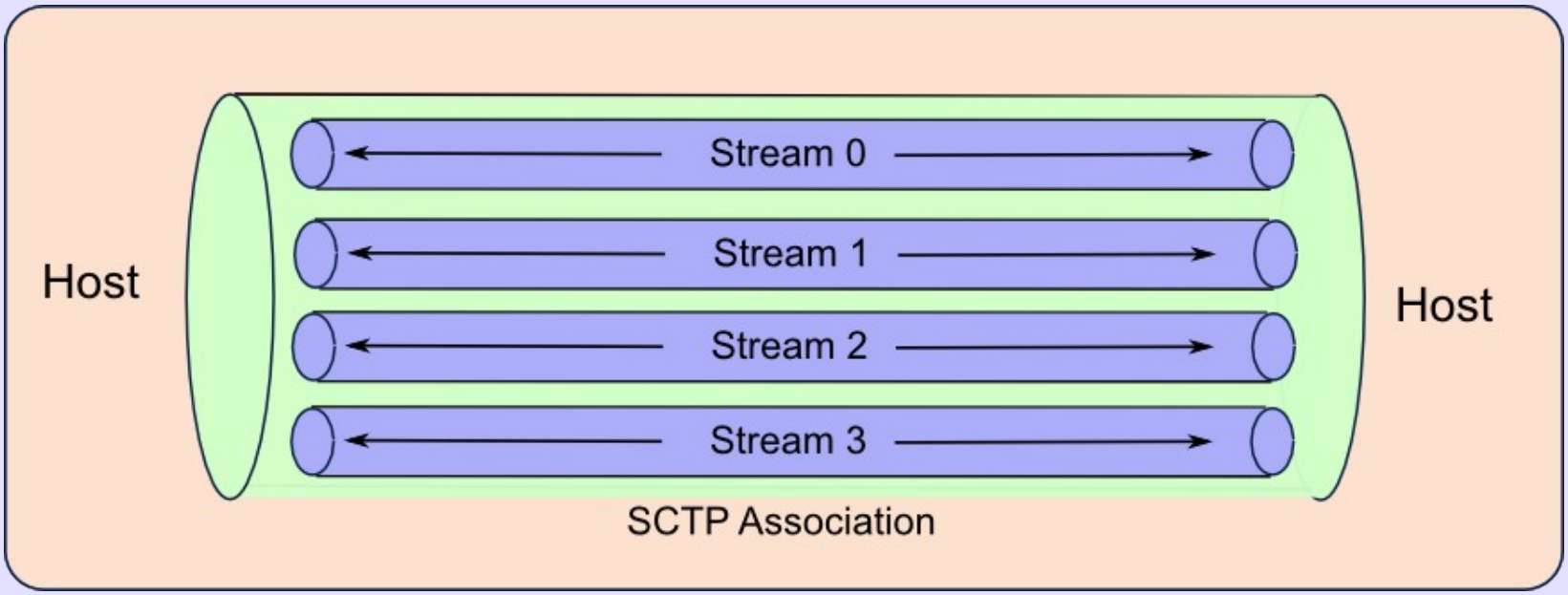
STCP/IP - Multi-homing



STCP/IP - Multi-homing

- SCTP monitors the paths of the association using a built-in heartbeat; upon detecting a path failure, the protocol sends traffic over the alternate path.
- It's not even necessary for the applications to know that a failover recovery occurred.

STCP/IP - Multi-Streaming



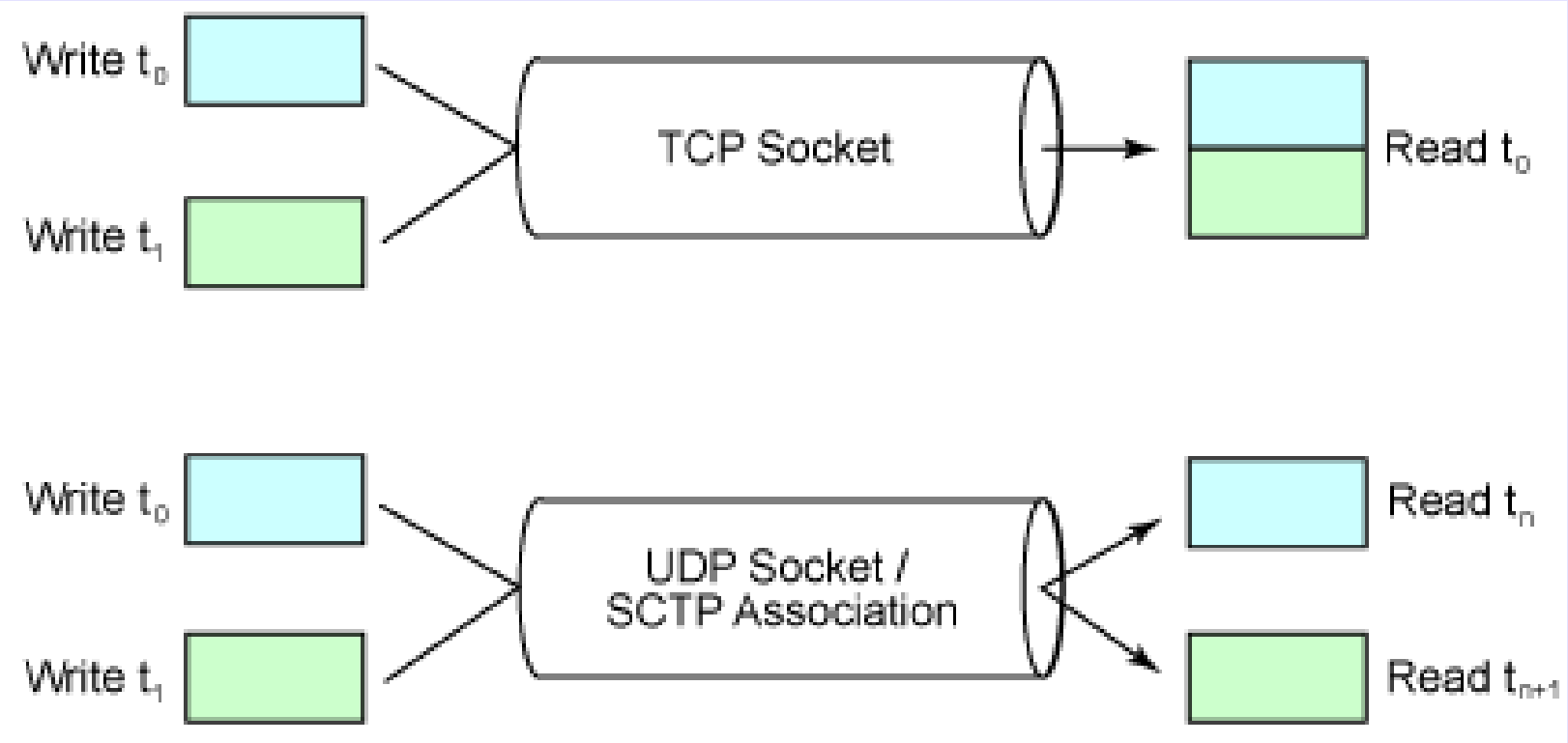
STCP/IP - Multi-Streaming

- In some ways, an SCTP association is like a TCP connection except that SCTP supports multiple streams within an association.
- All the streams within an association are independent but related to the association.
- Each stream has a stream number.

STCP/IP – Message Framing

- With message framing, the boundaries in which messages are communicated through a socket are preserved:
  - this means that if a client sends 100 bytes to a server followed by 50 bytes, the server will read 100 bytes and 50 bytes, respectively, for two reads. UDP also operates in this way, which makes it advantageous for message-oriented protocols.

STCP/IP – Message Framing



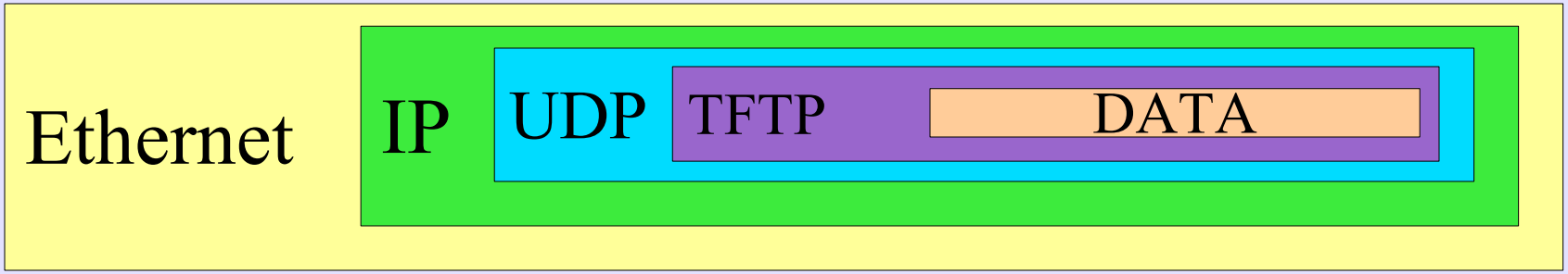
STCP/IP – Unordered Delivery

- Messages in SCTP are transferred reliably but not necessarily in the desired order.
- TCP guarantees that data is delivered in order (which is a good thing, considering TCP is a stream protocol).
- You can configure streams within SCTP to accept unordered messages if desired.

TCP – UDP - SCTP

Feature Name	UDP	TCP	SCTP
Connection oriented	No	Yes	Yes
Reliable transport	No	Yes	Yes
Preserve message boundary	Yes	No	Yes
Ordered delivery	No	Yes	Yes
Unordered delivery	Yes	No	Yes
Data checksum	Yes	Yes	Yes
Checksum size (bits)	16	16	32
Path MTU	No	Yes	Yes
Congestion control	No	Yes	Yes
Multiple streams	No	No	Yes
Multi-homing support	No	No	Yes
Bundling / Nagle	No	Yes	Yes

Data Encapsulation



ISO 7 Layer Model

- Application
- Presentation
- Session
- Transport
- Network
- Data Link
- Physical

Reality

- Application Layer (telnet, ftp, ssh, ...)
- Host-to-Host Transport (TCP, UDP, SCTP)
- Internet Layer (IP and Routing)
- Physical Network Layer (Ethernet, WIFI, DSL)

Socket Interface

- Network byte order - Big-Endian
- Host byte order
- Translation functions:
  - ntohl() of htonl()
  - ntohs() of htons()

struct sockaddr

```
struct sockaddr {
    unsigned short    sa_family;    // address family, AF_xxx
    char              sa_data[14];  // 14 bytes of protocol
address
};
```

struct sockaddr_in

```
/* Structure describing an Internet (IP) socket address. */
#define __SOCK_SIZE__   16          /* sizeof(struct sockaddr) */
struct sockaddr_in {
  sa_family_t           sin_family; /* Address family   */
  __be16                sin_port;   /* Port number      */
  struct in_addr        sin_addr;   /* Internet address */

  /* Pad to size of `struct sockaddr'. */
  unsigned char         __pad[__SOCK_SIZE__ - sizeof(short int) -
                        sizeof(unsigned short int) -
                        sizeof(struct in_addr)];
};
```

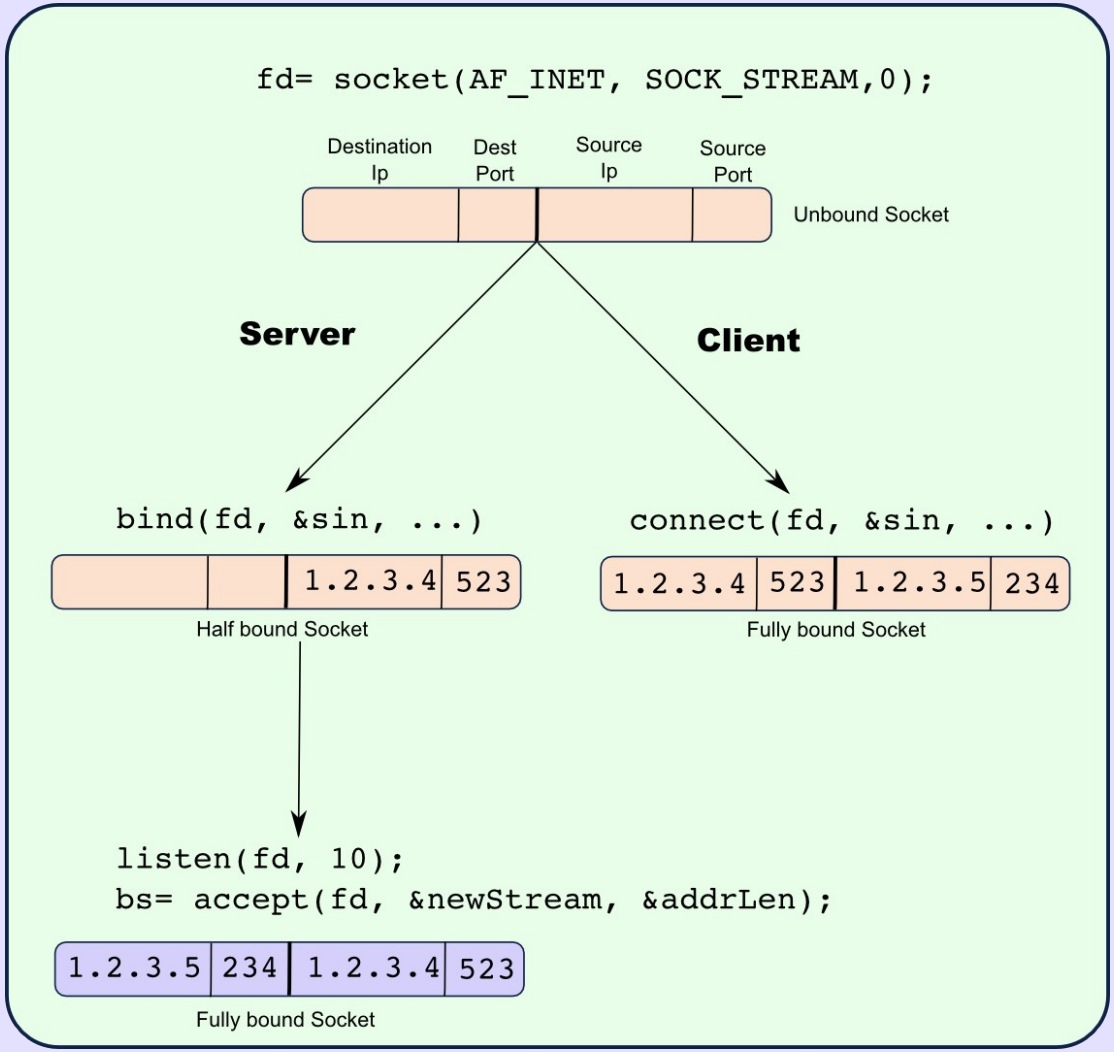
Network Addresses

```
sin.s_addr = inet_addr("10.12.110.57");
```

OR

```
if((hp = gethostbyname("lp")) == NULL) {
    fprintf(stderr,"server lp not found\n");
    exit(1);
}
bzero((char *)&sin, sizeof (sin));
bcopy(hp->h_addr, (char *)&sin.sin_addr,
    hp->h_length);
sin.sin_family= hp->h_addrtype; /* Type */
sin.sin_port = htons(9100);     /* Port number */
```

Socket – Server and Client



Unix Domain Socket

- When we want to connect to a socket on our local computer we can use a local socket rather than an internet domain socket.
- This saves us the overhead of TCP/IP for local transfers of data.

struct sockaddr_un

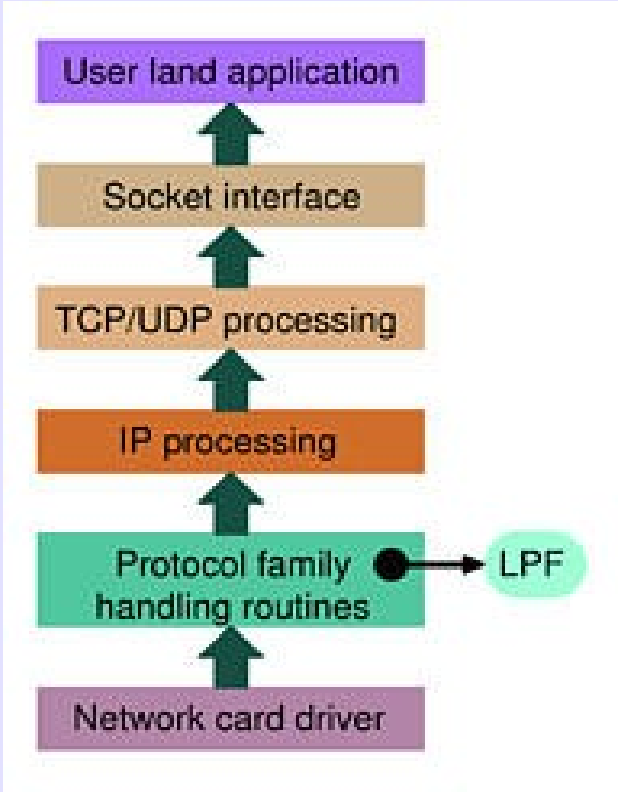
```
// Use socket(AF_UNIX, SOCK_STREAM, 0)


#define UNIX_PATH_MAX   108

struct sockaddr_un {
        sa_family_t sun_family; /* AF_UNIX */
        char sun_path[UNIX_PATH_MAX];   /* pathname */
};
```

PF_PACKET - Packet Interface on Device Level

- Packet sockets are used to receive or send raw packets at the device driver (OSI Layer 2) level.
- The PF_PACKET family supports two slightly different socket types, SOCK_DGRAM (no MAC header) and SOCK_RAW (with MAC header).

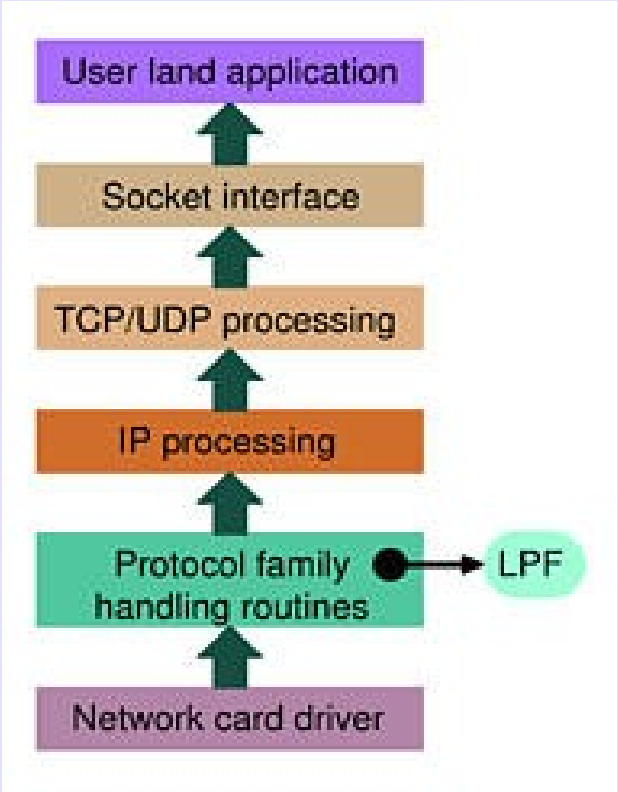
Linux Packet Filter - LPF

In order to be as flexible as possible, and not to limit the programmer to a set of predefined conditions, the packet-filtering engine is actually implemented as a state machine running a user-defined program.



Linux Packet Filter - LPF

- The program is written in a specific pseudo-machine code language called BPF (Berkeley packet filter).
- BPF actually looks like an assembly language that works on the packet data.



Tcpdump Program

- The tcpdump program can be used to filter packets and easily write bpf programs.

```
# /usr/sbin/tcpdump -d src 1.2.3.4
(000) ldh      [12]
(001) jeq      #0x800           jt 2    jf 4
(002) ld       [26]
(003) jeq      #0x1020304       jt 8    jf 9
(004) jeq      #0x806           jt 6    jf 5
(005) jeq      #0x8035          jt 6    jf 9
(006) ld       [28]
(007) jeq      #0x1020304       jt 8    jf 9
(008) ret      #96
(009) ret      #0
```

In The Beginning, there was BSD

- The standard BSD networking programs, “route”, “ifconfig” and “netstat” are now over 20 years old.
- Routes are added with the ioctl call with the SIOCADDRT argument.

Iproute2 and Linux 2.2

- The Linus routing code was rewritten for the 2.2 series kernel.
- The user level interface to the kernel's networking was changed by the introduction of the “netlink” IP service protocol.

```
netlink_socket = socket(PF_NETLINK,
        socket_type, netlink_family);
```

The ip Command

- The “ip” command take the place of most of the traditional BSD networking utilities and more.
- It uses the netlink IP service protocol in order to interface with the kernel.
- Rfc3549, “Linux Netlink as an IP Services Protocol” defines this network service.

ip -- command syntax.

Generic form of ip command is:

ip [OPTIONS] OBJECT [COMMAND
[ARGUMENTS]]

where OPTIONS is set of optional modifiers affecting general behaviour of the ip utility or changing its output. All the options begin with character '-' and may be used both in long and abbreviated form.

Object Types

- link -- network device.
- address -- protocol (IP or IPv6) address on a device.
- neighbour -- ARP or NDISC cache entry.
- route -- routing table entry.
- rule -- rule in routing policy database.
- mroute -- multicast routing entry.
- tunnel -- tunnel over IP.

ip link -- Network Device Configuration

- ip link set -- change device attributes.
  - ip link set dummy address 00:00:00:00:00:01
    - change station address of the interface dummy.
  - ip link set dummy up
    - start the interface dummy.
- ip link show -- look at device attributes
  - ip link ls eth0
    - 3: eth0: <BROADCAST,MULTICAST,UP> mtu 1500 qdisc cbq qlen 100 link/ether 00:a0:cc:66:18:78 brd ff:ff:ff:ff:ff:ff

ip address -- Protocol Address Management

- ip address add -- add new protocol address
  - ip addr add 127.0.0.1/8 dev lo brd + scope host
    - add usual loopback address to loopback device.
  - ip addr a 10.0.0.1/24 brd + dev eth0 label eth0:Alias
    - add address 10.0.0.1 with prefix length 24 (i.e. netmask 255.255.255.0, standard broadcast and label eth0:Alias to the interface eth0.

ip address -- Protocol Address Management

- ip address delete -- delete protocol address
  - ip addr del 127.0.0.1/8 dev lo
    - deletes loopback address from loopback device. It would be better not to try to repeat this experiment.

ip address -- Protocol Address Management

- ip address show -- protocol addresses
  - ip addr ls eth0
    - 3: eth0: <BROADCAST,MULTICAST,UP> mtu 1500 qdisc cbq qlen 100
    - link/ether 00:a0:cc:66:18:78 brd ff:ff:ff:ff:ff:ff
    - inet 193.233.7.90/24 brd 193.233.7.255 scope global eth0
    - inet6 3ffe:2400:0:1:2a0:ccff:fe66:1878/64 scope global dynamic
    - valid_lft forever preferred_lft 604746sec
    - net6 fe80::2a0:ccff:fe66:1878/10 scope link

ip address flush -- Flush Protocol Addresses

- ip -s -s a f to 10/8
  - 2: dummy   inet 10.7.7.7/16 brd 10.7.255.255 scope global dummy
  - 3: eth0   inet 10.10.7.7/16 brd 10.10.255.255 scope global eth0
  - 4: eth1   inet 10.8.7.7/16 brd 10.8.255.255 scope global eth1
  - *** Round 1, deleting 3 addresses ***
  - *** Flush is complete after 1 round *

ip neighbour -- neighbour/arp tables management

- These commands take the functionality of the BSD 'arp” command.
  - ip neighbour add -- add new neighbour entry
  - ip neighbour change -- change existing entry
  - ip neighbour replace -- add new or change existing one.

ip route -- Routing Table Management

- ip route add -- add new route
- ip route change -- change route
- ip route replace -- change route or add new one
  - ip route add 10.0.0/24 via 193.233.7.65
    - add plain route to network 10.0.0/24 via gateway 193.233.7.65
  - ip route add nat 192.203.80.142 via 193.233.7.83
    - announce that address 192.203.80.144 is not real one, but should be translated to 193.233.7.83 before forwarding

ip route show -- list routes

- ip ro ls 193.233.7/24
  - 193.233.7.0/24 dev eth0 proto gated/conn scope link src 193.233.7.65 realms inr.ac
- ip ro ls 193.233.7.82 tab cache
  - 193.233.7.82 from 193.233.7.82 dev eth0 src 193.233.7.65 realms inr.ac/inr.ac cache <src-direct,redirect> mtu 1500 rtt 300 iif eth0
  - 193.233.7.82 dev eth0 src 193.233.7.65 realms inr.ac cache mtu 1500 rtt 300

ip route flush -- flush routing tables

- ip -6 -s -s ro flush cache
  - 3ffe:2400::220:afff:fef4:c5d1 via 3ffe:2400::220:afff:fef4:c5d1 dev eth0 metric 0 cache used 2 age 12sec mtu 1500 rtt 300
  - 3ffe:2400::280:adff:feb7:8034 via 3ffe:2400::280:adff:feb7:8034 dev eth0 metric 0
  - ....
  - *** Round 1, deleting 6 entries ***
  - *** Flush is complete after 1 round ***

ip route get -- get single route

- ip route get 193.233.7.82
  - 193.233.7.82 dev eth0 src 193.233.7.65 realms inr.ac cache mtu 1500 rtt 300
- ip r g 193.233.7.82 from 193.233.7.82 iif eth0
  - 193.233.7.82 from 193.233.7.82 dev eth0 src 193.233.7.65 realms inr.ac/inr.ac cache <src-direct,redirect> mtu 1500 rtt 300 iif eth0
- ip r g 224.2.127.254 from 193.233.7.82 iif eth0
  - multicast 224.2.127.254 from 193.233.7.82 dev lo src 193.233.7.65 realms inr.ac/cosmos cache <mc> iif eth0 Oifs: eth1 pimreg

ip rule -- routing policy database management

- Classic routing algorithms used in the Internet make routing decisions based only on the destination address of packets.
- In some circumstances we want to route packets differently depending not only on the destination addresses, but also on another packet fields: source address, IP protocol, transport protocol ports or even packet payload. This task is called ``policy routing''.

ip rule -- routing policy database management

- To solve this task conventional destination based routing table, ordered according to the longest match rule, is replaced with ``routing policy database'' (or RPDB), which selects route executing some set of rules.
- The rules may have lots of keys of different nature and therefore they have no natural ordering, but imposed by administrator.

ip rule -- routing policy database management

- ip ru add from 192.203.80.0/24 table inr.ruhep prio 220
  - Route packets with source addresses from 192.203.80/24 according to routing table inr.ruhep.
- ip ru add from 193.233.7.83 nat 192.203.80.144 table 1 prio 320
  - Translate packet source 193.233.7.83 to 192.203.80.144 and route it according to table #1.

ip rule show -- list rules

- ip ru ls
  - 0: from all lookup local
  - 200: from 192.203.80.0/24 to 193.233.7.0/24 lookup main
  - 210: from 192.203.80.0/24 to 192.203.80.0/24 lookup main
  - 220: from 192.203.80.0/24 lookup inr.ruhep realms inr.ruhep/radio-msu
  - 300: from 193.233.7.83 to 193.233.7.0/24 lookup main
  - 32766: from all lookup main

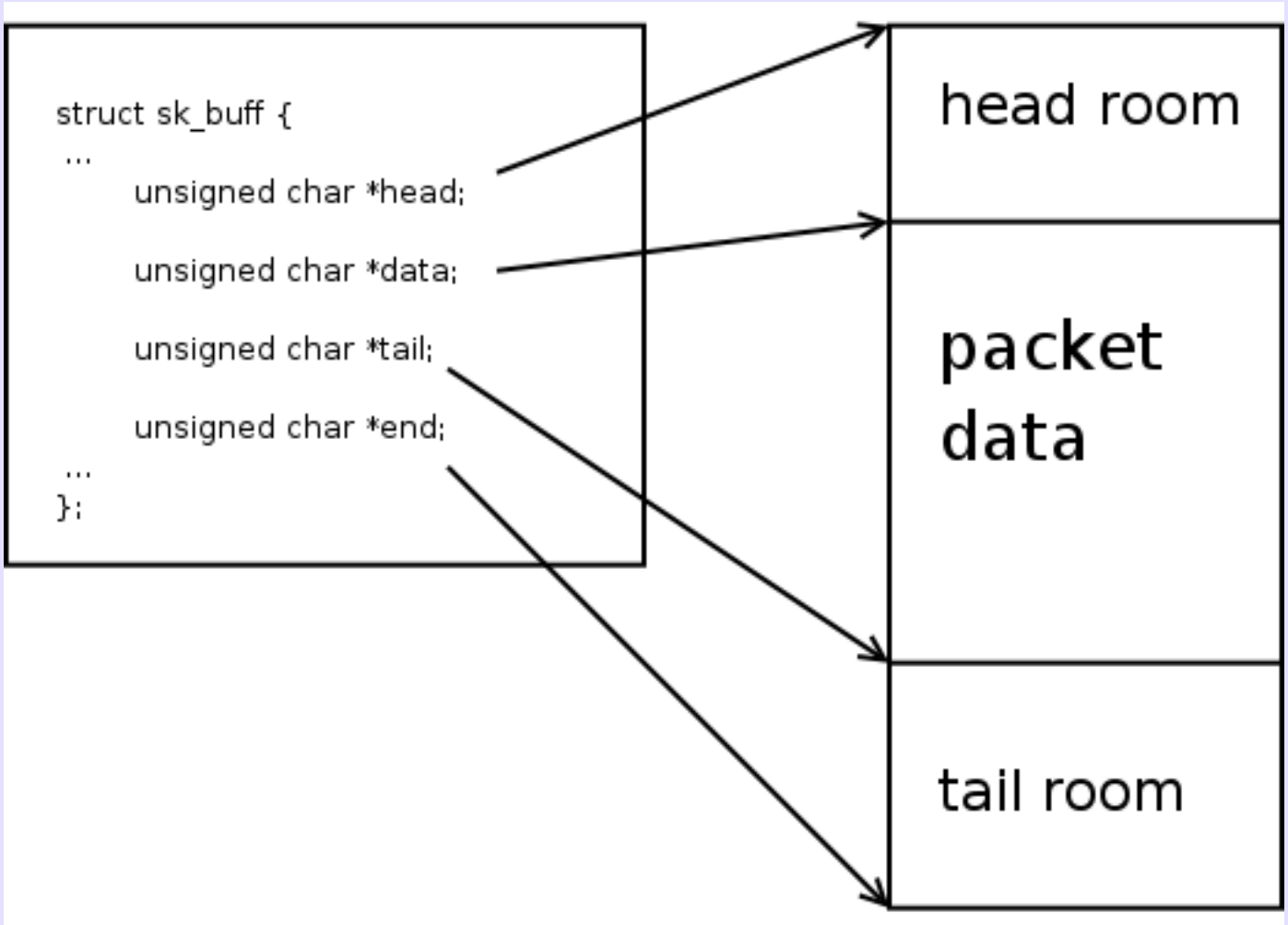
ip monitor -- state monitoring

- Netlink can be used to dynamically monitor changes in routes.
- As routing changes are made these changes are reported to via netlink via a monitoring socket.
- This allows applications to monitor the state of the routing tables
- The ip utility allows to monitor state of devices, addresses and routes continuously.

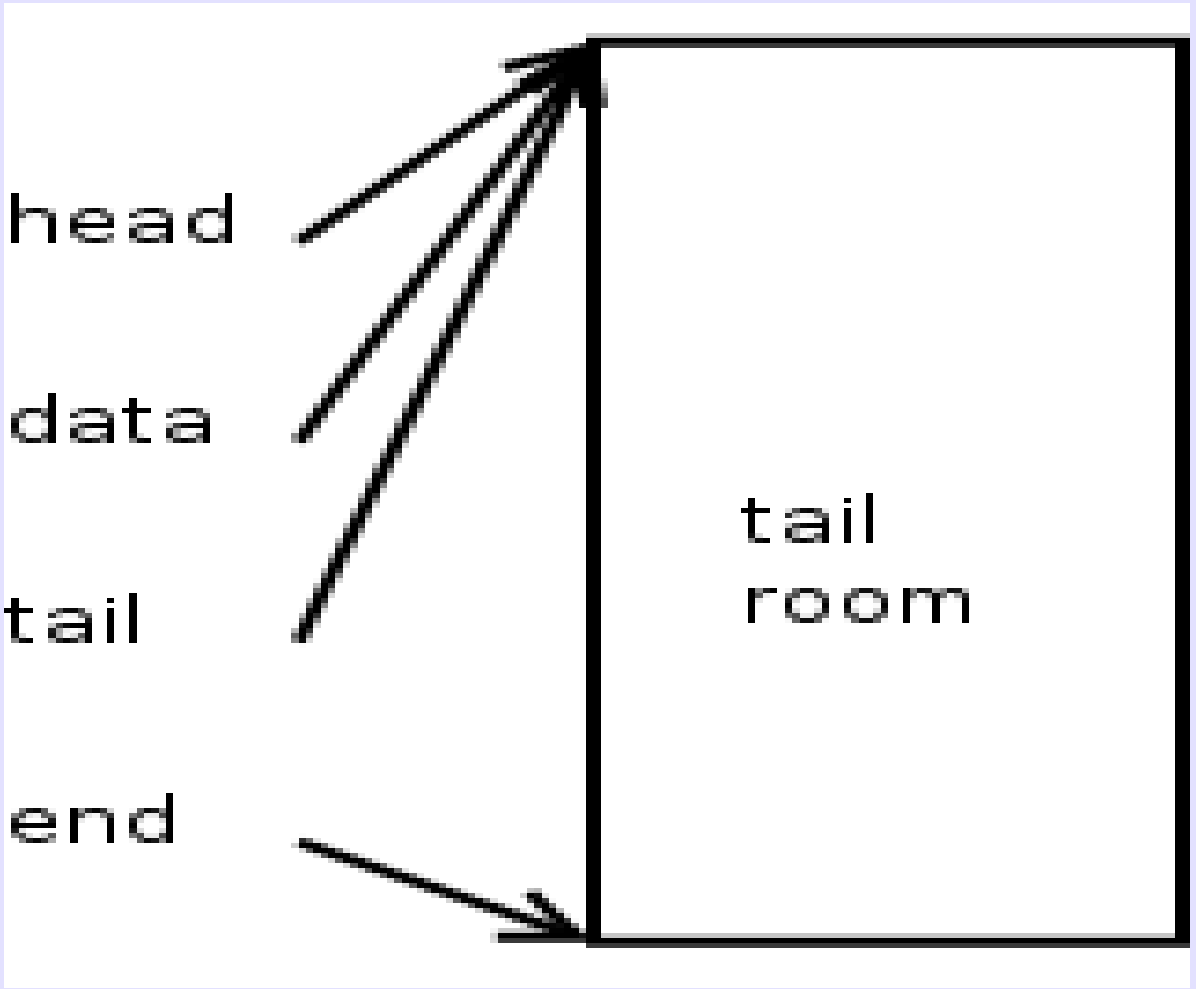
Network Device Drivers

- We will examine the loopback device driver as the simplest example of a Linux network driver.
- We first have to understand the important kernel structure the sk_buff.

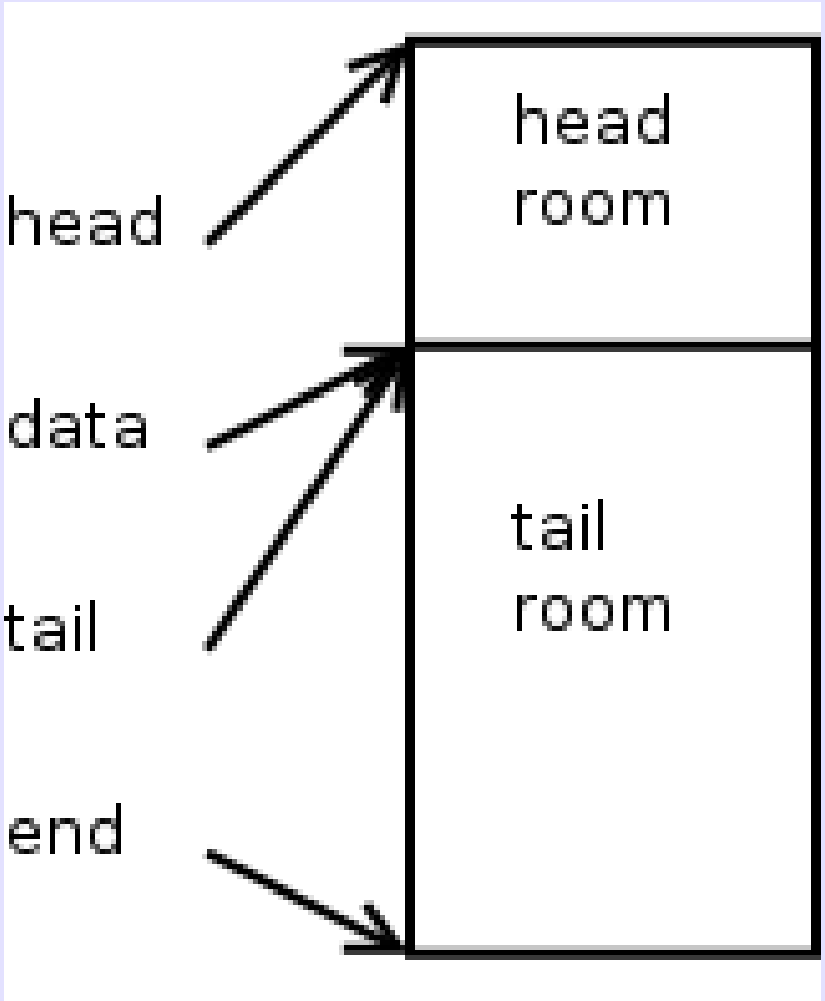
The sk_buff


struct sk_buff {
...
unsigned char *head;
unsigned char *data;
unsigned char *tail;
unsigned char *end;
...
};
head room
packet data
tail room

After alloc_skb()



After skb_reserve()



After alloc_put() and copy of data