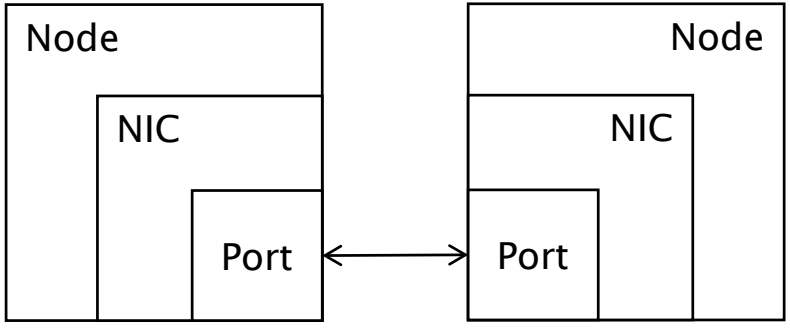
Life of a packet

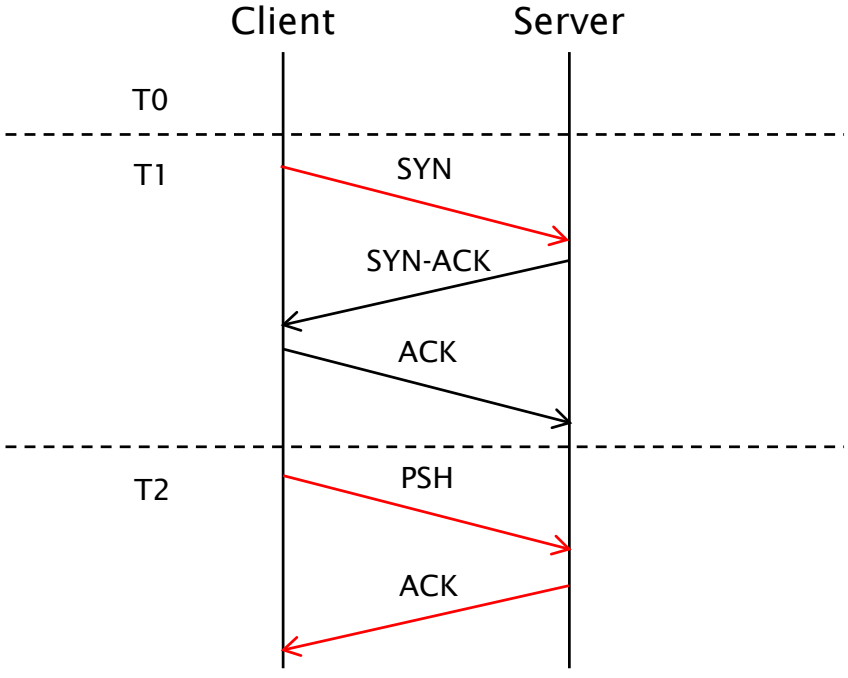
Xiao Jia

Pygmal Technologies

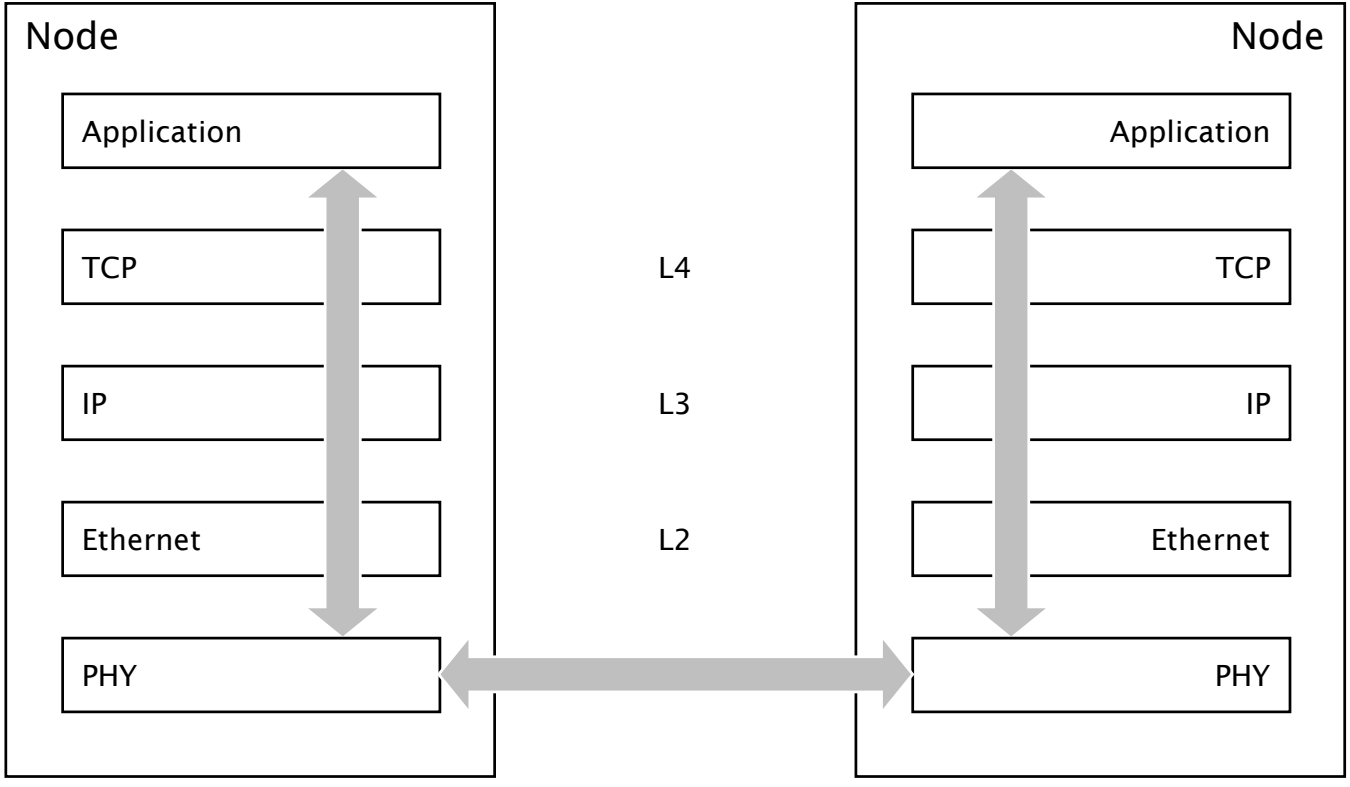
Scope



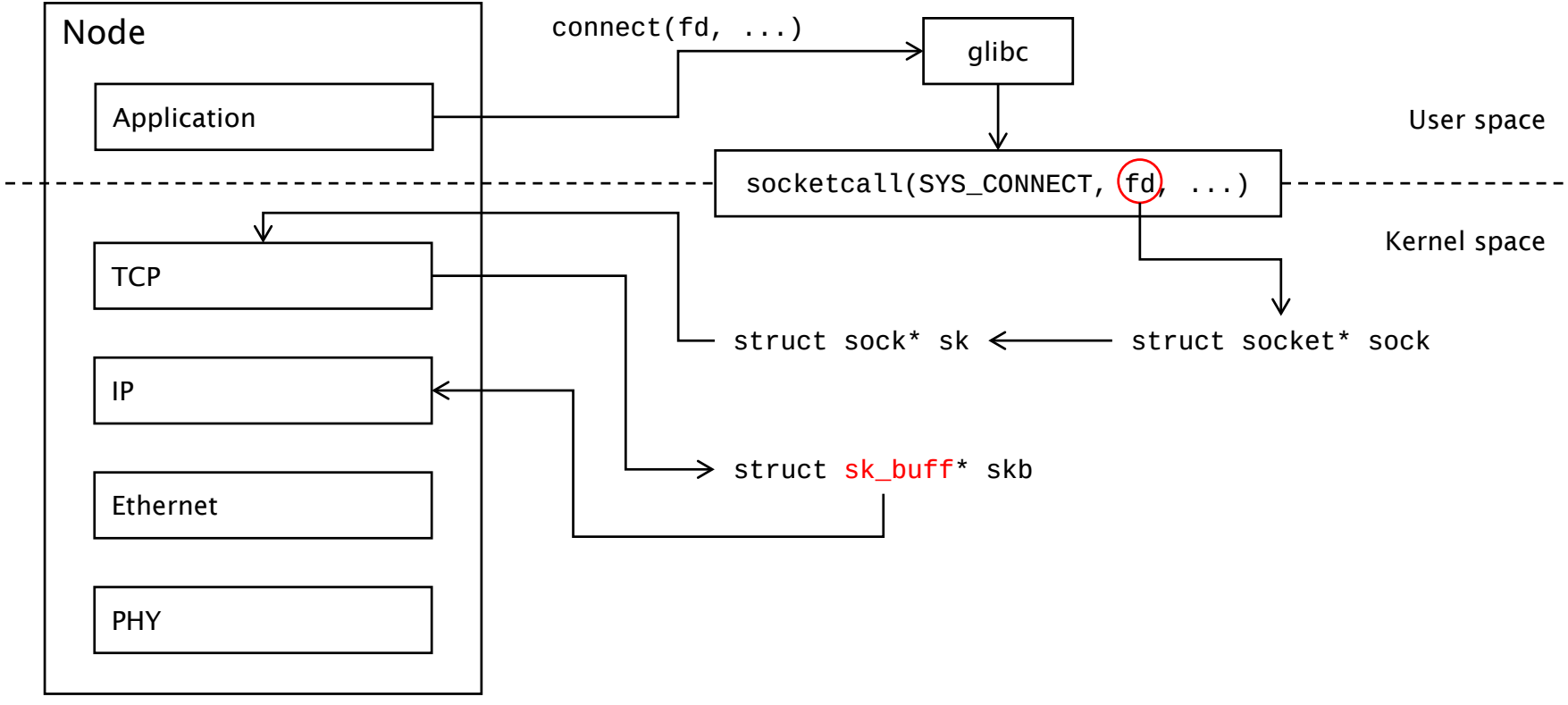
Time	Client	Server
T0		bind + listen
T1	connect	accept
T2	write	read



Layers


Node
Application
TCP
IP
Ethernet
PHY
L4
L3
L2
Node
Application
TCP
IP
Ethernet
PHY

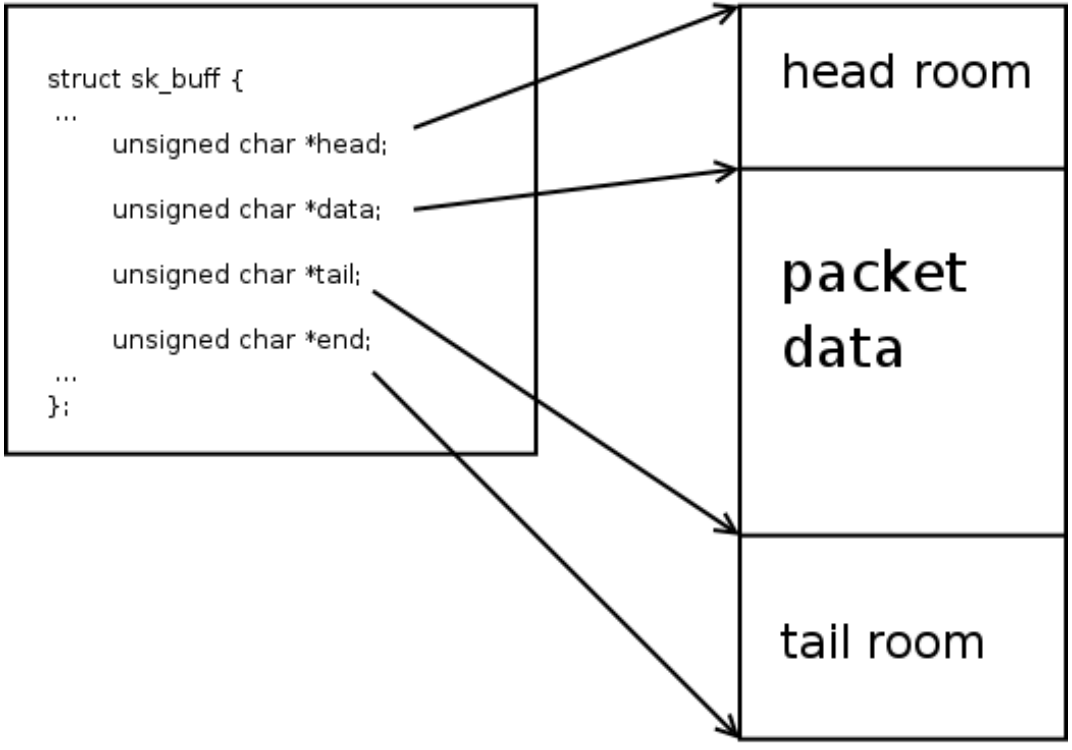
User/kernel space transition



SKB (1)

```
struct sk_buff {
        struct sk_buff          *next;
        struct sk_buff          *prev;
        struct sk_buff_head     *list;

        struct sock             *sk;            for e.g. proper memory accounting

        struct net_device       *dev;

        struct dst_entry        *dst;

        char                    cb[40];         Control block.  TCP uses this to store
                                                seq. number and retransmission state.
```

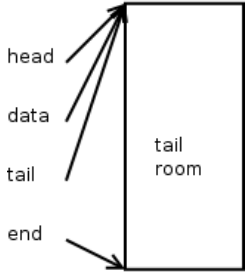
SKB data area (1)



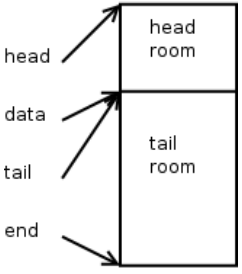
Source: http://vger.kernel.org/~davem/skb_data.html

SKB data area (2)

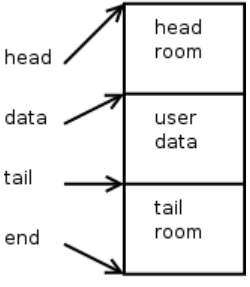
```
skb = alloc_skb(len, GFP_KERNEL);
```

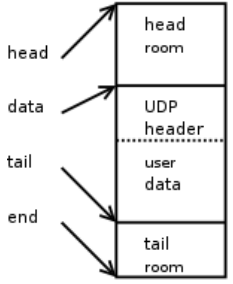



```
skb_reserve(skb, header_len);
```

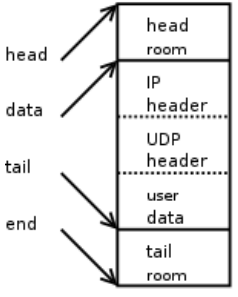



```
unsigned char *data = skb_put(skb, user_data_len);
int err = 0;
skb->csum = csum_and_copy_from_user(user_pointer, data,
                                    user_data_len, 0, &err);
if (err)
        goto user_fault;
```






```
struct inet_sock *inet = inet_sk(sk);
struct flowi *fl = &inet->cork.fl;
struct udphdr *uh;

skb->h.raw = skb_push(skb, sizeof(struct udphdr));
uh = skb->h.uh
uh->source = fl->fl_ip_sport;
uh->dest = fl->fl_ip_dport;
uh->len = htons(user_data_len);
uh->check = 0;
skb->csum = csum_partial((char *)uh,
                         sizeof(struct udphdr), skb->csum);
uh->check = csum_tcpudp_magic(fl->fl4_src, fl->fl4_dst,
                              user_data_len, IPPROTO_UDP, skb->csum);
if (uh->check == 0)
        uh->check = -1;
```




```
struct rtable *rt = inet->cork.rt;
struct iphdr *iph;

skb->nh.raw = skb_push(skb, sizeof(struct iphdr));
iph = skb->nh.iph;
iph->version = 4;
iph->ihl = 5;
iph->tos = inet->tos;
iph->tot_len = htons(skb->len);
iph->frag_off = 0;
iph->id = htons(inet->id++);
iph->ttl = ip_select_ttl(inet, &rt->u.dst);
iph->protocol = sk->sk_protocol; /* IPPROTO_UDP in this case */
iph->saddr = rt->rt_src;
iph->daddr = rt->rt_dst;
ip_send_check(iph);

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

SKB (2)

```
struct sk_buff {

        union {
                struct tcphdr    *th;
                struct udphdr    *uh;
                struct icmphdr   *icmph;
                struct igmphdr   *igmph;
                struct iphdr     *ipiph;
                struct ipv6hdr   *ipv6h;
                unsigned char    *raw;
        } h;

        union {
                struct iphdr     *iph;
                struct ipv6hdr   *ipv6h;
                struct arphdr    *arph;
                unsigned char    *raw;
        } nh;

        union {
                unsigned char    *raw;
        } mac;
```

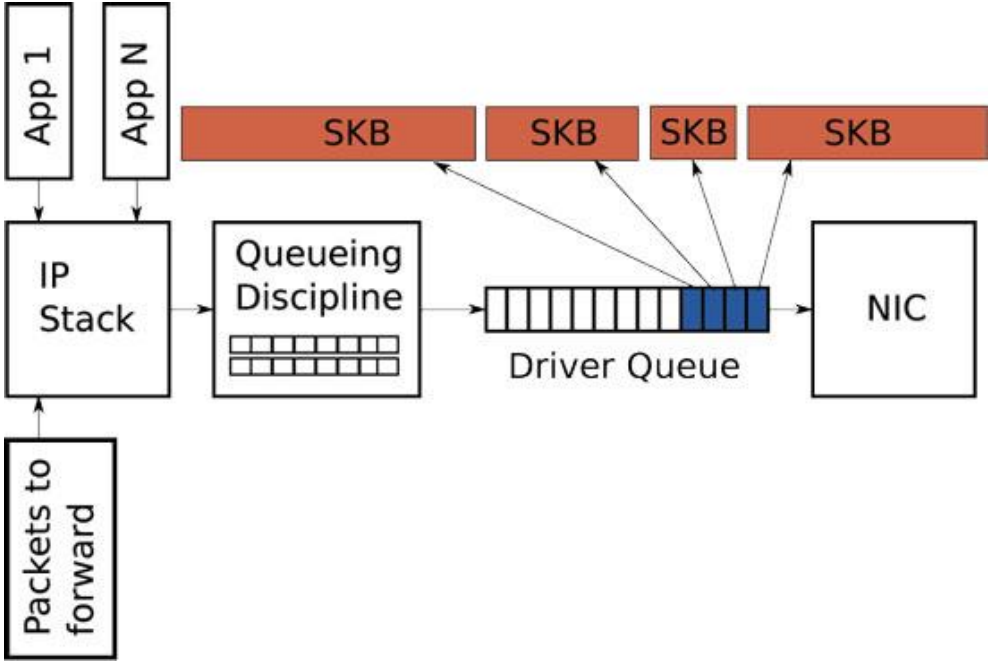
Transmission path

- write/sendto/sendmsg
- tcp_sendmsg
- tcp_transmit_skb
- ip_queue_xmit
- ip_output
- dev_queue_xmit
- netif_schedule
- dev_hard_start_xmit
- dev_kfree_skb_irq

Receive path

- netif_rx_schedule
- net_rx_action
- netif_receive_skb
- ip_rcv
- ip_local_deliver
- tcp_v4_rcv
- tcp_rcv_established
- skb_copy_datagram_iovec
- read/recvfrom/recvmsg

Queueing



Source: http://www.linuxjournal.com/content/queueing-linux-network-stack

Debugging tools

- netperf
- google/neper
- perf
- tcpdump
- wireshark
- strace
- ethtool
- kprobe
- ftrace

Missing pieces (a lot of them)

- qdisc
- netfilter (iptables)
- GRO
- RFS
- NAPI
- bonding
- Protocol details (e.g. TCP retransmission)
- ... ...