CSCI 6907.11

Adv. Net. Sys. Prog.

Lecture 10 - Network Layers

Tim Wood
CS@GWU
2015

Some content from Kurose and Ross

How was using GENI?

Class projects?

Today

Layers

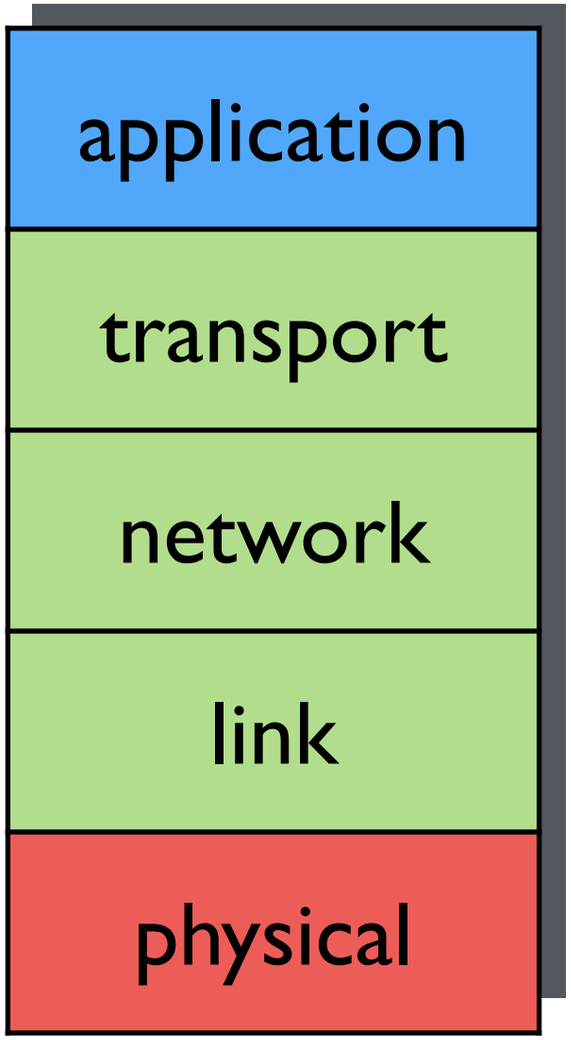
- Protocols
- Packets
- Operating Systems
- Sockets
- Applications

Speak up!

Internet protocol stack

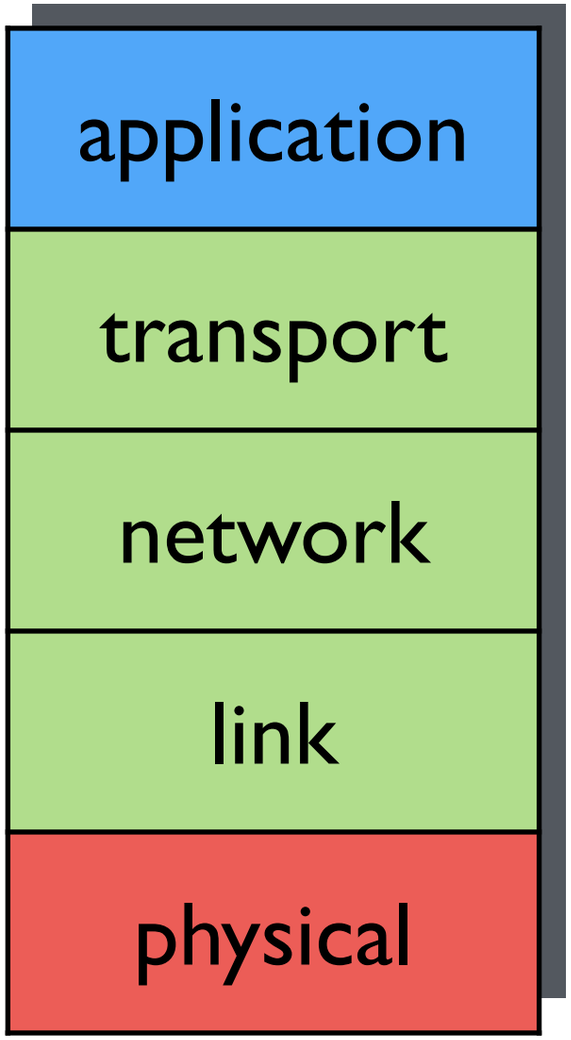
What do these layers do?

Why do we have them?



Internet protocol stack

- *application:* supporting network applications
  - FTP, SMTP, HTTP
- *transport:* process-process data transfer
  - TCP, UDP
- *network:* routing of datagrams from source to destination
  - IP, routing protocols
- *link:* data transfer between neighboring network elements
  - Ethernet, 802.111 (WiFi), PPP
- *physical:* bits "on the wire"



Ethernet Frame

Preamble

Src/Dest MAC

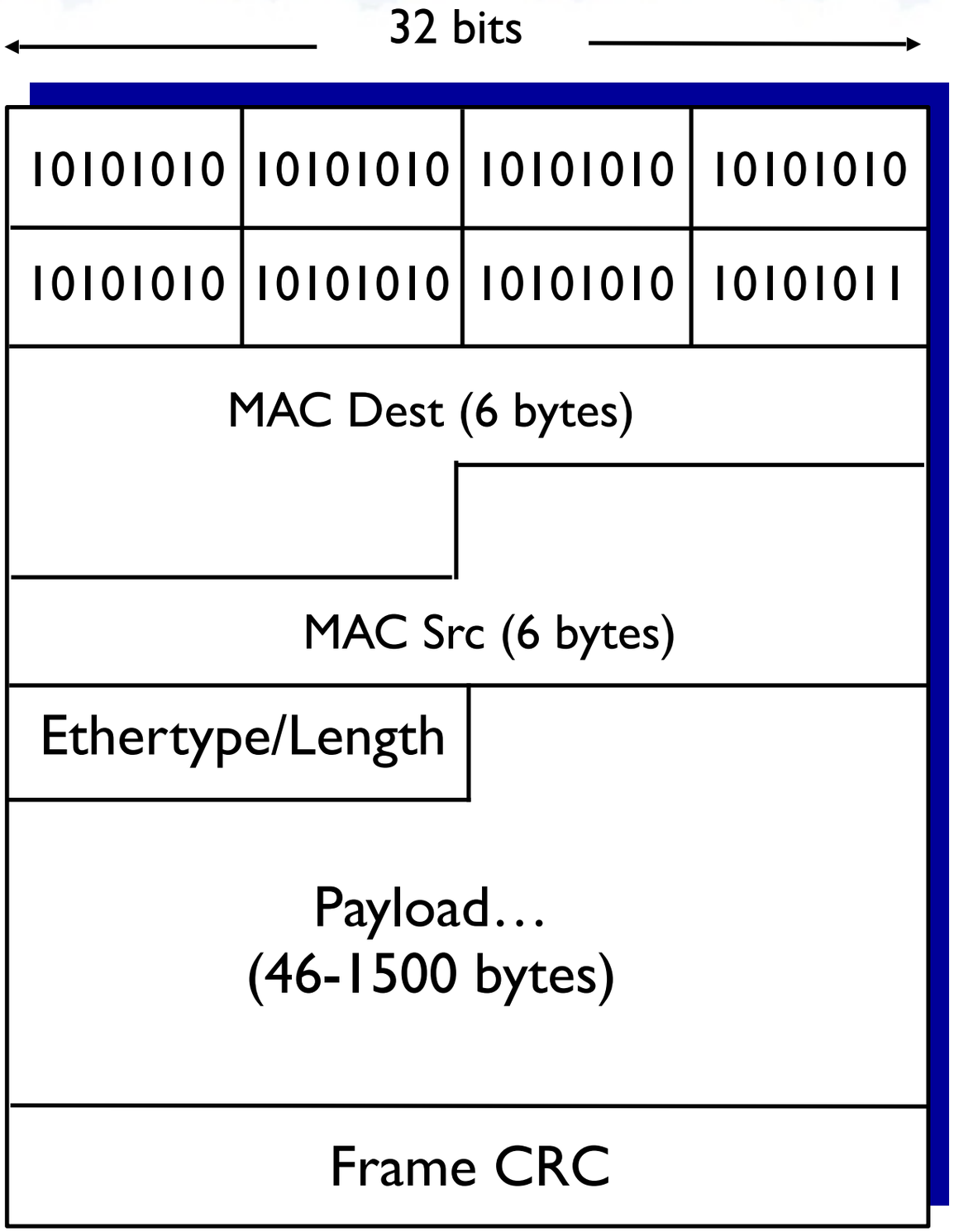
Length

Payload

CRC

Interpacket gap

What's with the preamble?



IP datagram format

Network layer

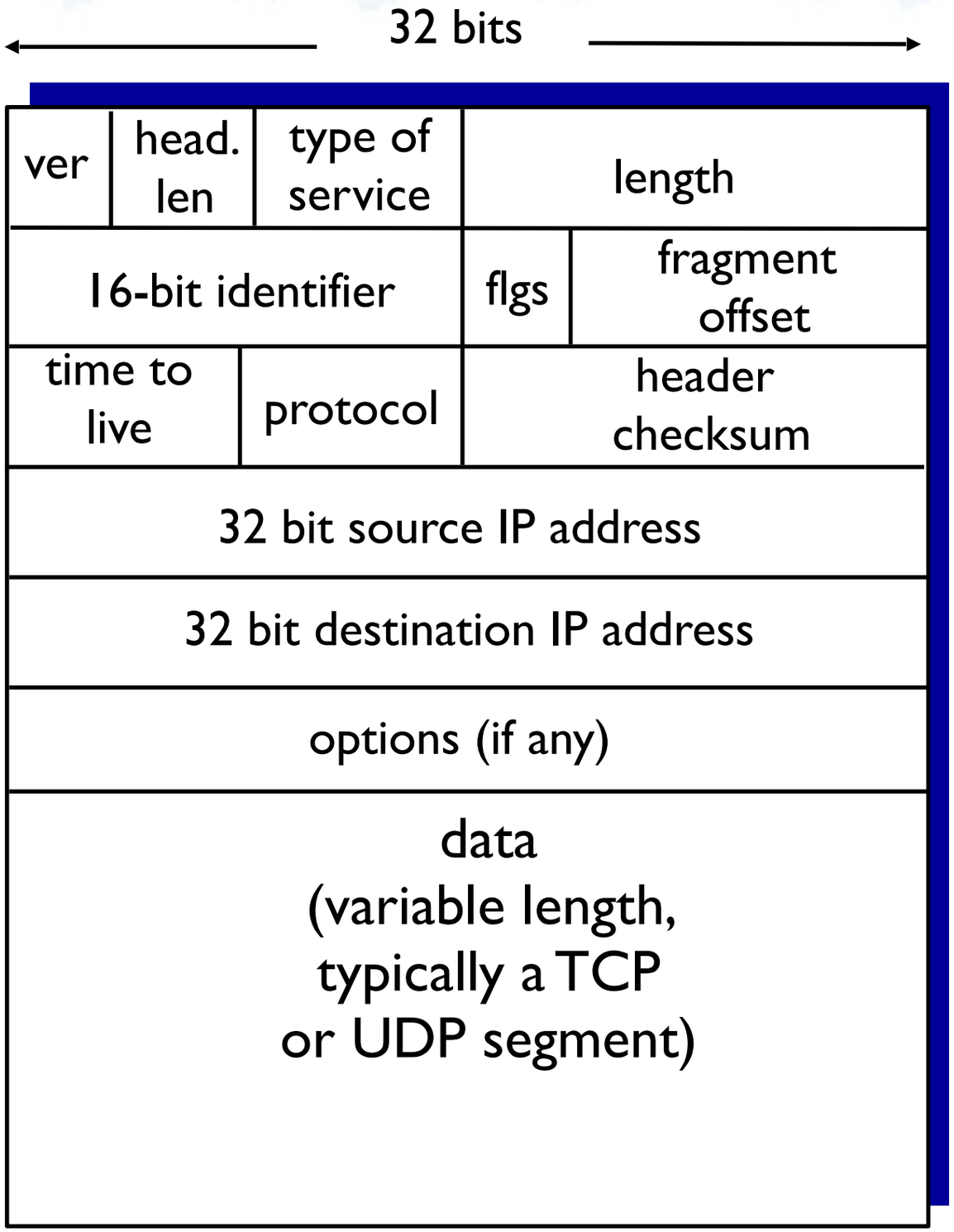
- Determines routing through the network

Source and destination

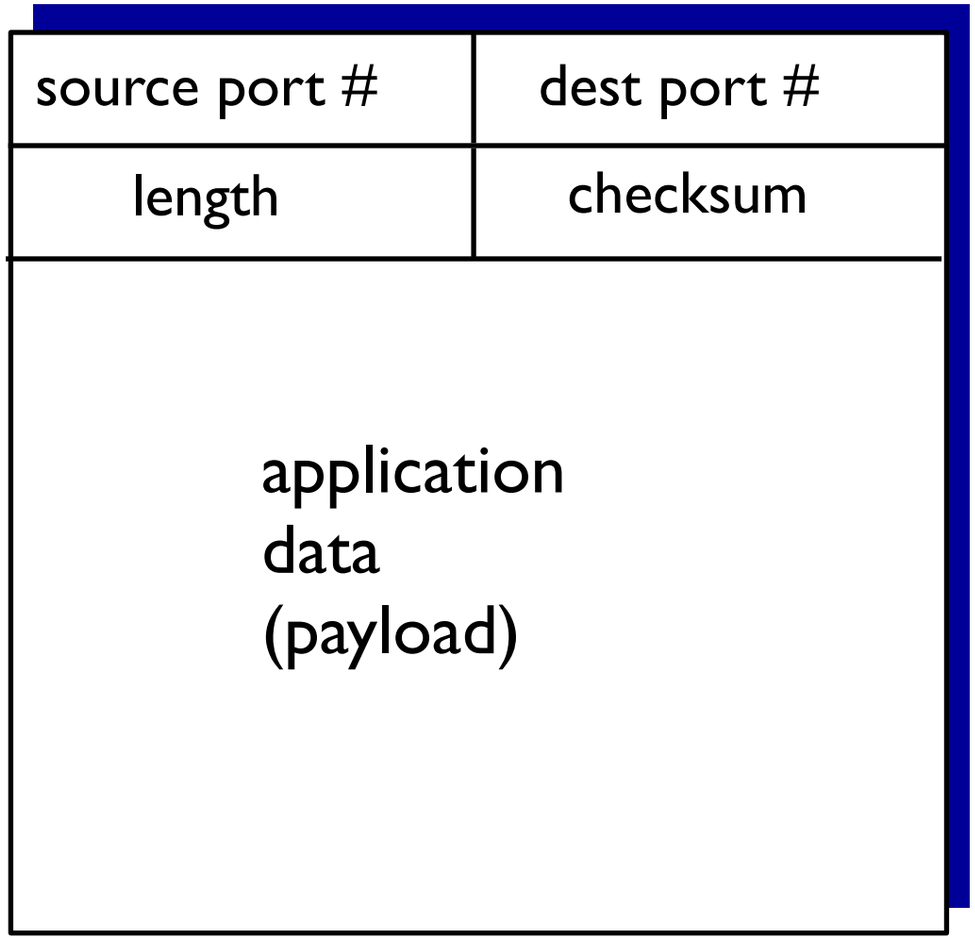
- IP address

Protocol field specifies the next type of header

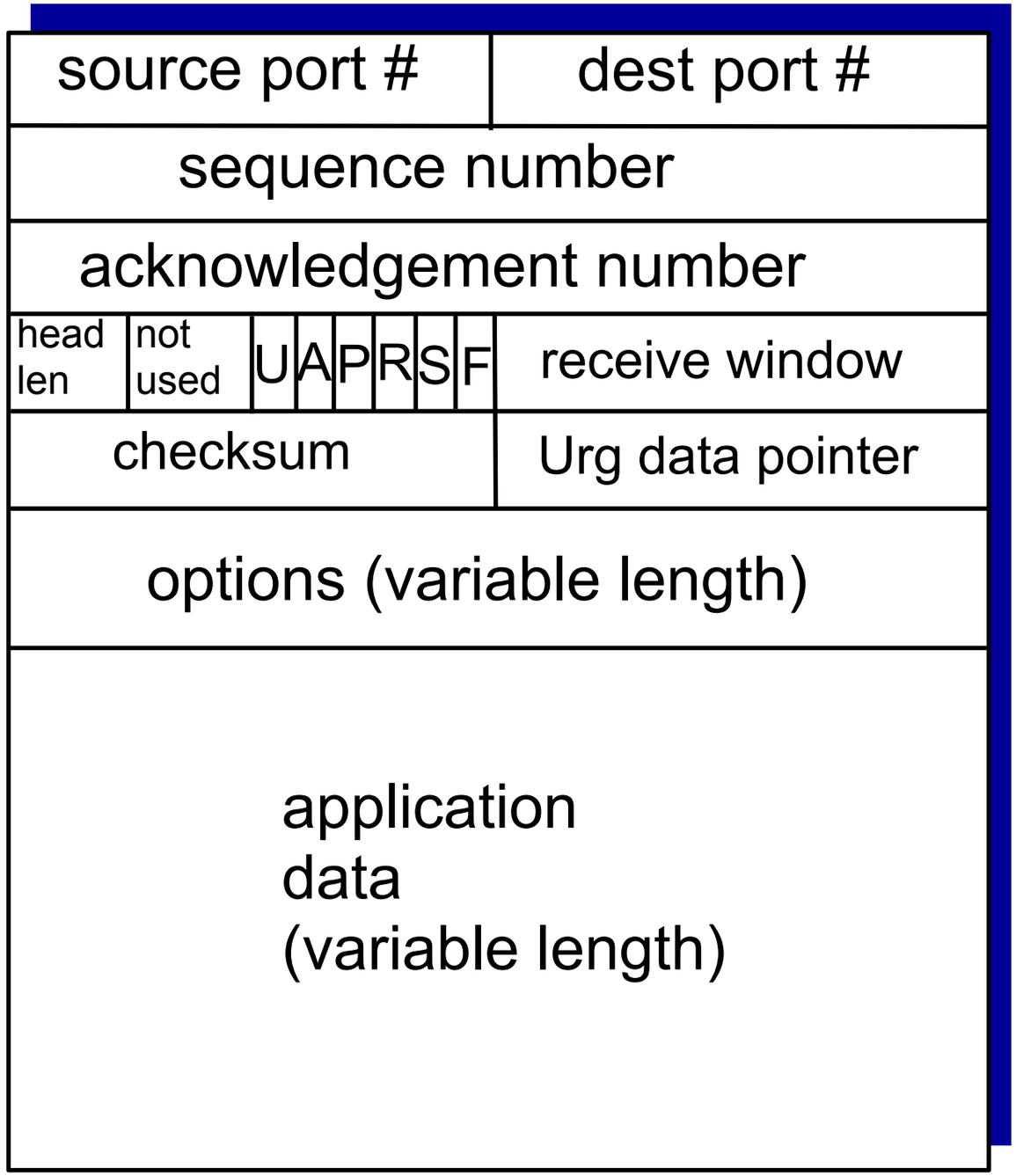
Why do we use IPs? Why not just use MAC addresses?



TCP and UDP



UDP format

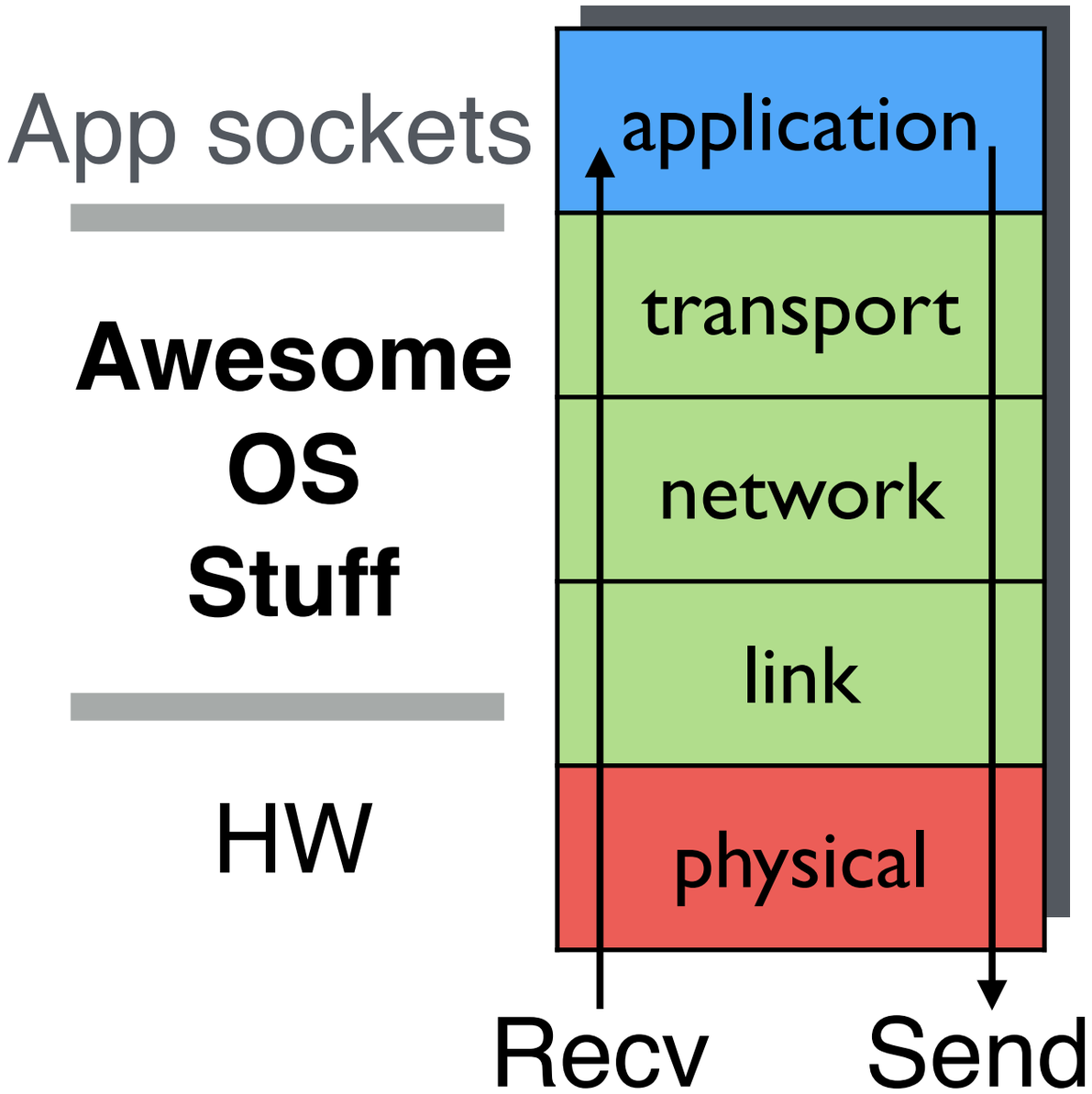


TCP format

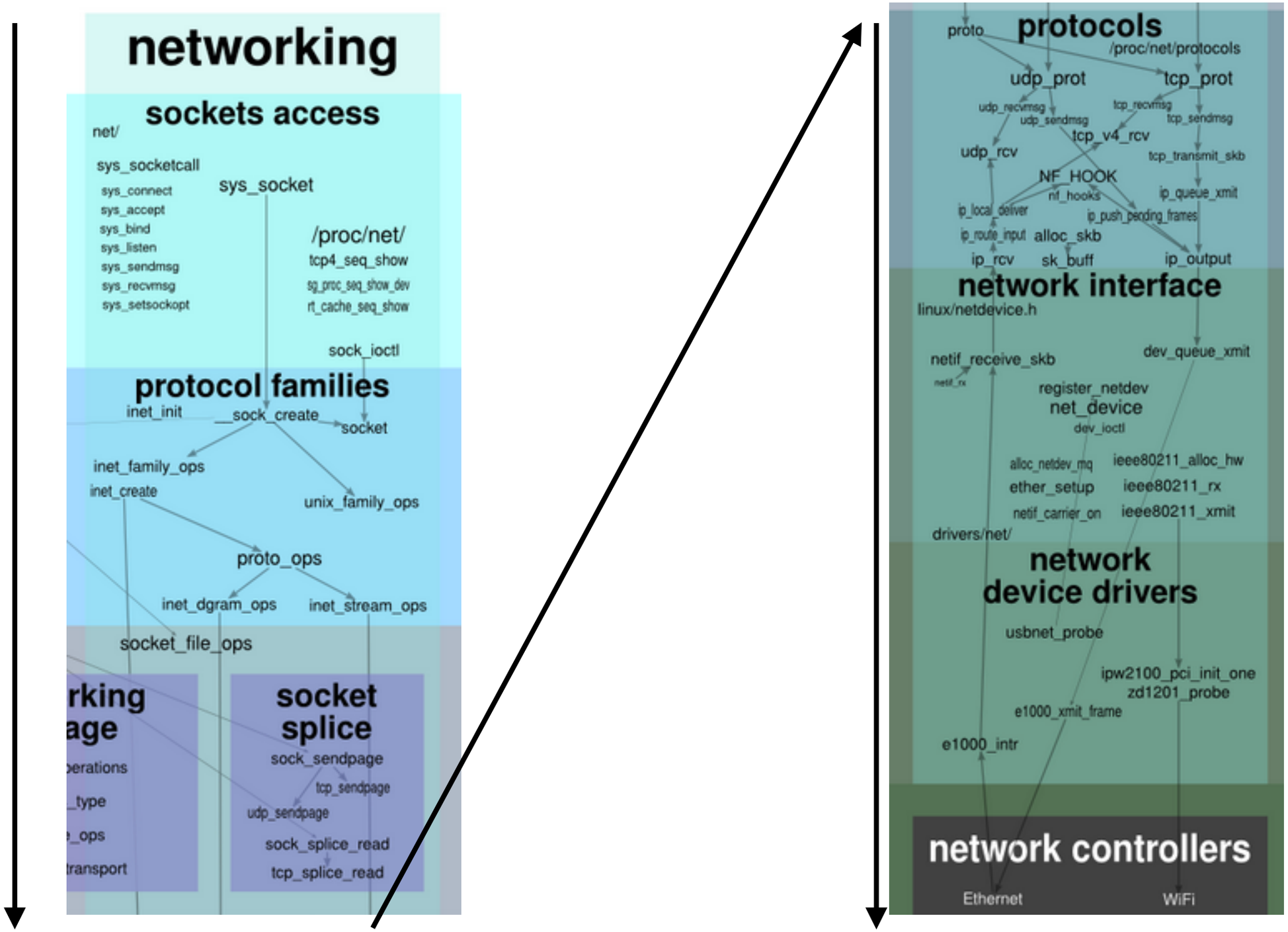
Why have both UDP and TCP?

What’s it all look like together?

Packet Path


App sockets
Awesome OS Stuff
HW
application
transport
network
link
physical
Recv
Send

Linux Network Stack



Receiving a Packet

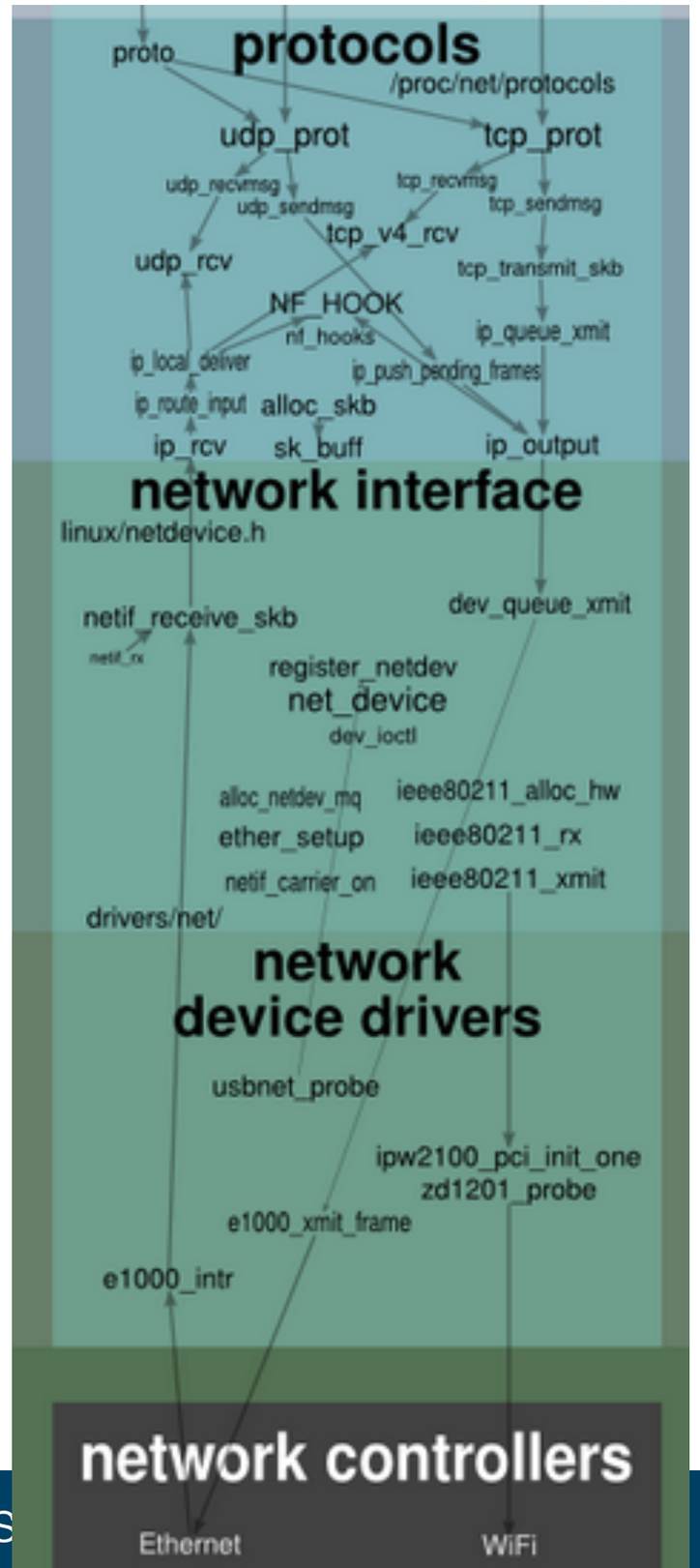
NIC registers an interrupt handler with the OS

- Handler is called when packet arrives
- Packet is copied into kernel memory

Handler uses an **sk_buff** to refer to the packet

Calls **ip_recv()**

- processes IP header
- Determines if packet is local or to be forwarded



Receiving a Packet

Determine packet's transport protocol

- UDP, TCP, etc

Match the UDP/TCP packet to the correct socket

- Use destination port number

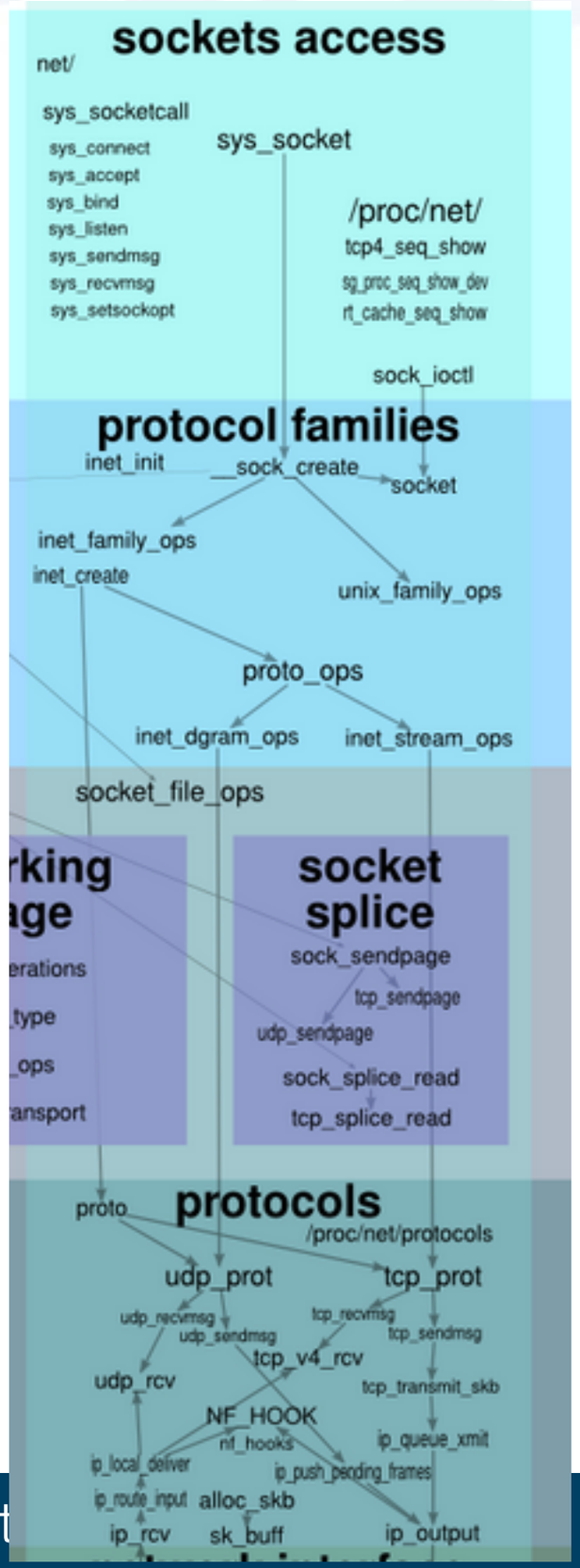
Hold onto the packet

- until sometime later…

User app calls **recv()**

- Kernel calls **copy_to_user()**

Data is available to application



How do we send a packet from a socket?

Send a packet

Open a socket
- ARP lookup for IP

Send library call
- write system call
- copy to buffer in kernel

Transport layer:
- Fill in TCP/UDP header

Network layer:
- Route lookup, fill in IP

Link layer:
- MAC lookup

Packet prep and DMA

File	Function/description	time ns	delta ns
user program	`sendto` system call	8	96
uipc_syscalls.c	`sys_sendto`	104	
uipc_syscalls.c	`sendit`	111	
uipc_syscalls.c	`kern_sendit`	118	
uipc_socket.c	`sosend`	—	
uipc_socket.c	`sosend_dgram` sockbuf locking, mbuf allocation, copyin	146	137
udp_usrreq.c	`udp_send`	273	
udp_usrreq.c	`udp_output`	273	57
ip_output.c	`ip_output` route lookup, ip header setup	330	198
if_ethersubr.c	`ether_output` MAC header lookup and copy, loopback	528	162
if_ethersubr.c	`ether_output_frame`	690	
ixgbe.c	`ixgbe_mq_start`	698	
ixgbe.c	`ixgbe_mq_start_locked`	720	
ixgbe.c	`ixgbe_xmit` mbuf mangling, device programming	730	220
–	on wire	950	

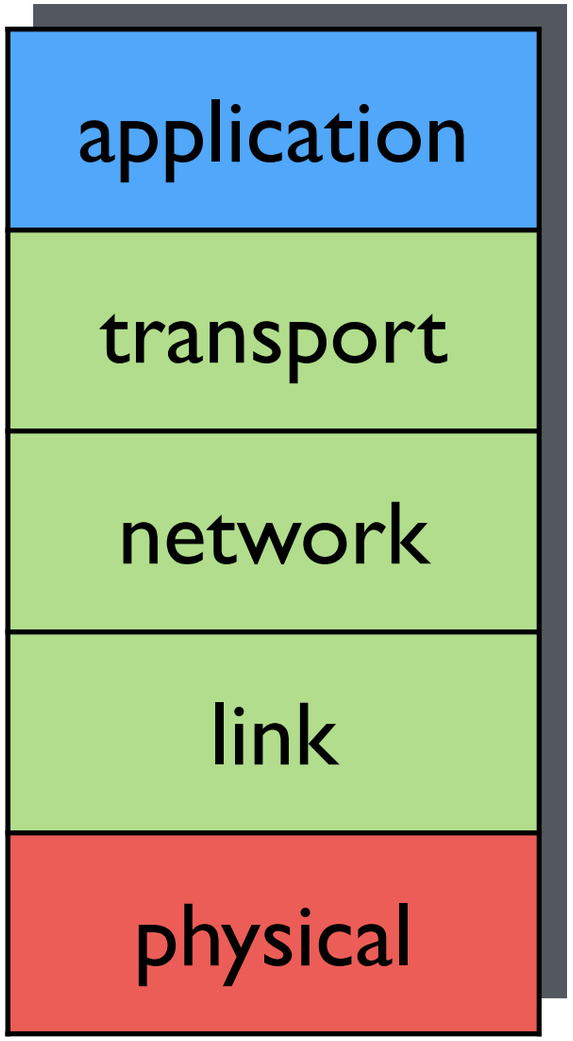
Figure 2: The path and execution times for `sendto()` on a recent FreeBSD HEAD 64-bit, i7-870 at 2.93 GHz + Turboboost, Intel 10 Gbit NIC and ixgbe driver. Measurements done with a single process issuing sendto() calls. Values have a 5% tolerance and are averaged over multiple 5s tests.

source: netmap @ Usenix ATC 12

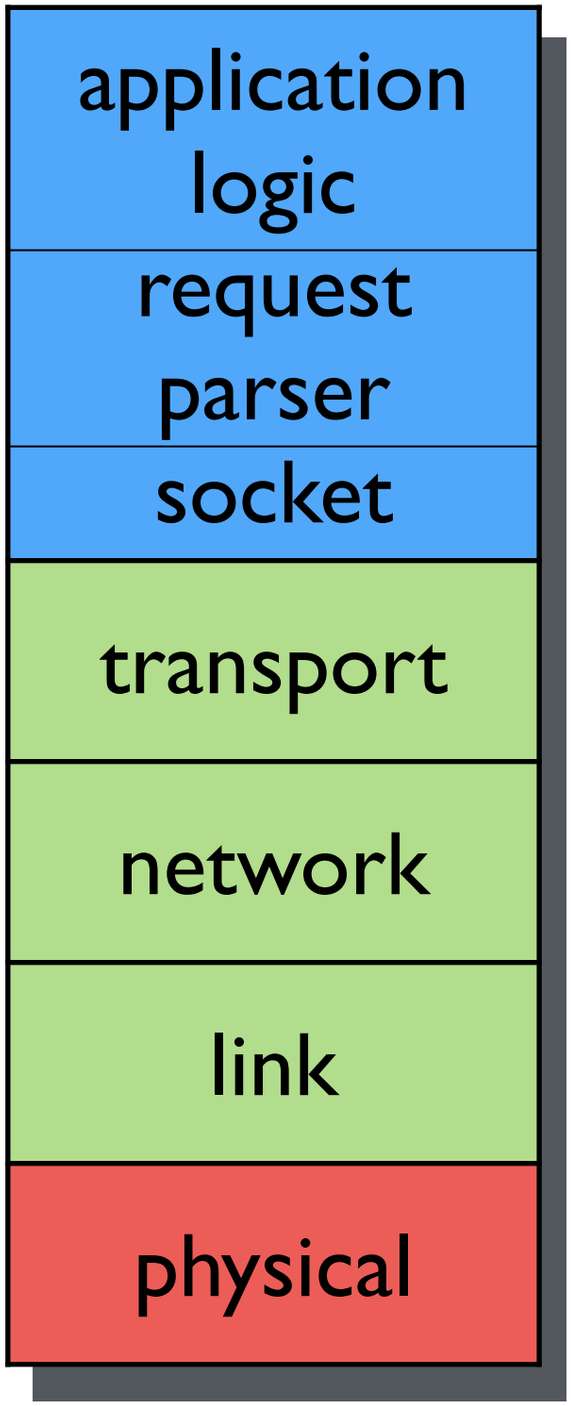
What happens once it reaches the socket and beyond?

Consider a web server, for example…

Layering…



More layering!



Back to work!

We have lots of **issues**

- Bugs on prior assignments
- POX SDN examples

Check for bugs on your prior work

Pick at least one SDN issue to solve

Use good git habits!

- Branch per issue
- Clean push requests
- Follow formatting guides!