Networking in Linux

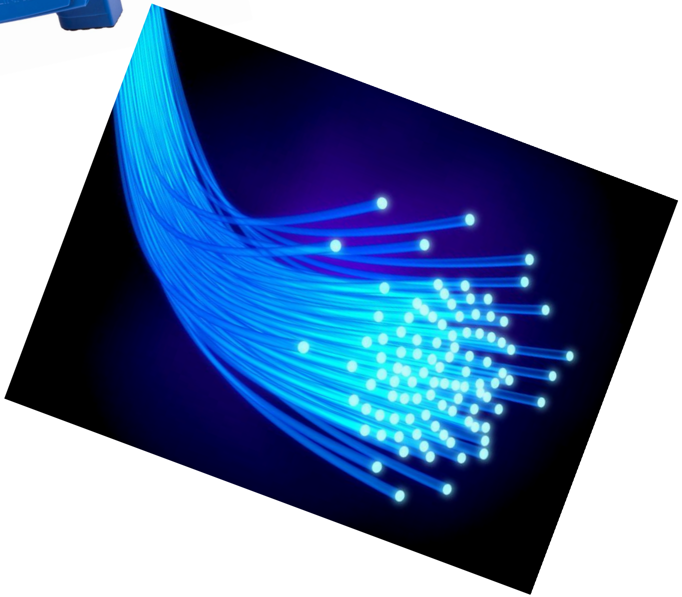
Neel Shah

What is a network?

- A bunch of computers that can exchange information between each other and/or share resources with each other.


NETGEAR

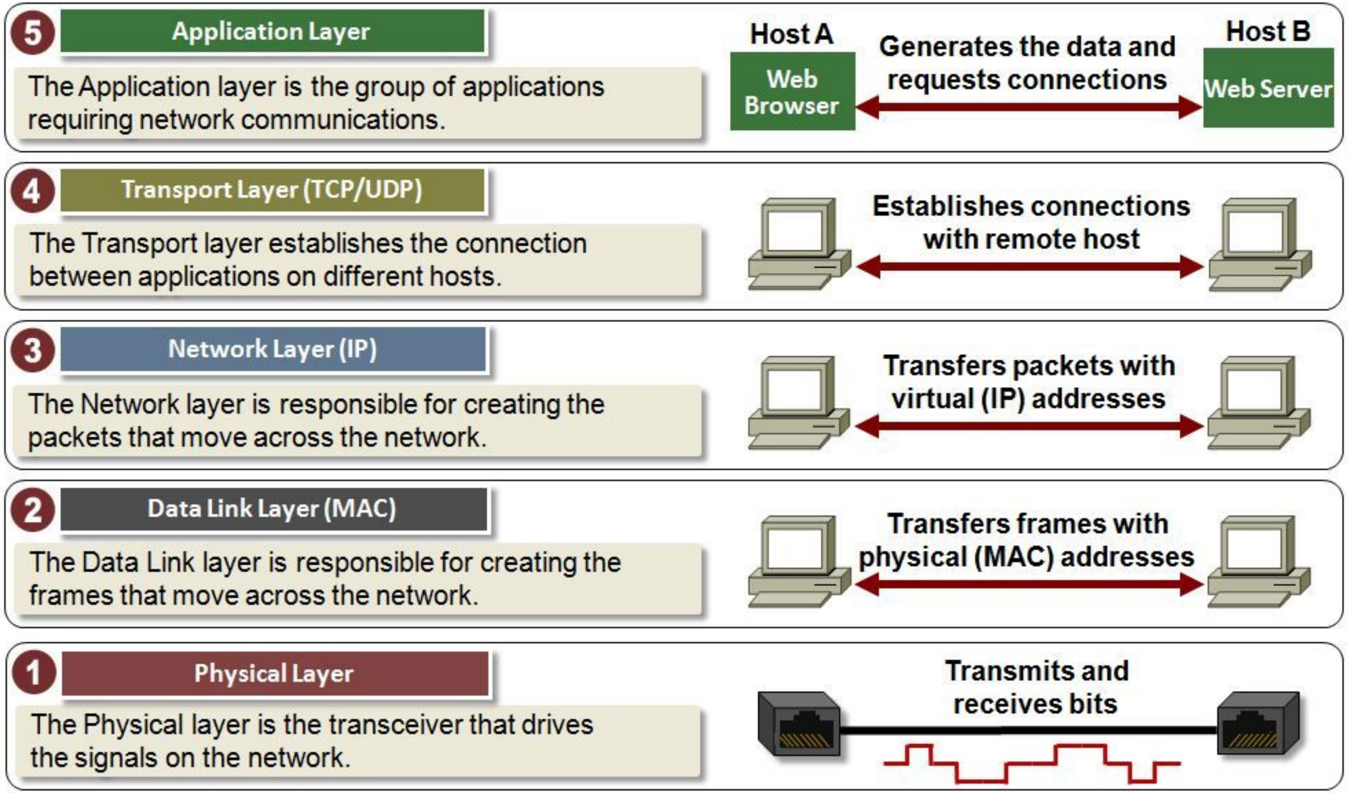

LINKSYS



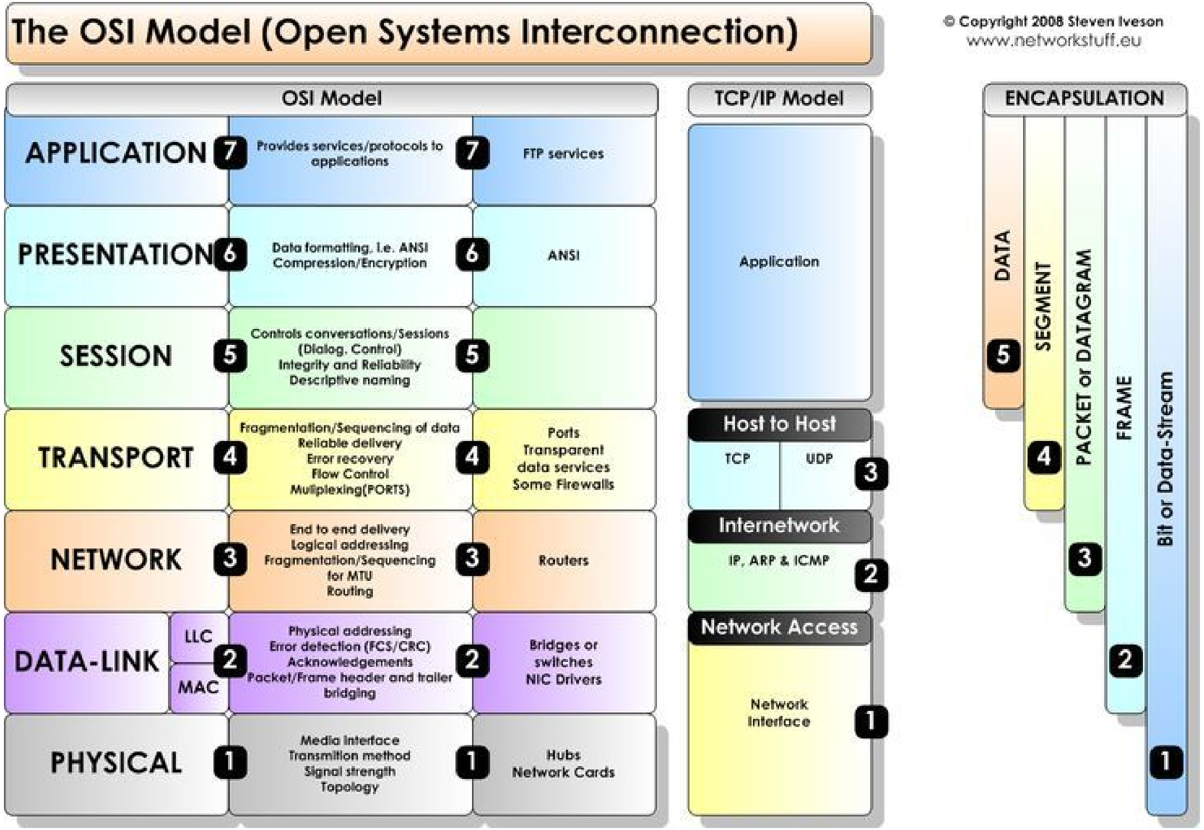
There's a stack! (7 Layer)

Layer	Function	Example
Application (7)	Services that are used with end user applications	SMTP,
Presentation (6)	Formats the data so that it can be viewed by the user Encrypt and decrypt	JPG, GIF, HTTPS, SSL, TLS
Session (5)	Establishes/ends connections between two hosts	NetBIOS, PPTP
Transport (4)	Responsible for the transport protocol and error handling	TCP, UDP
Network (3)	Reads the IP address form the packet.	Routers, Layer 3 Switches
Data Link (2)	Reads the MAC address from the data packet	Switches
Physical (1)	Send data on to the physical wire.	Hubs, NICS, Cable

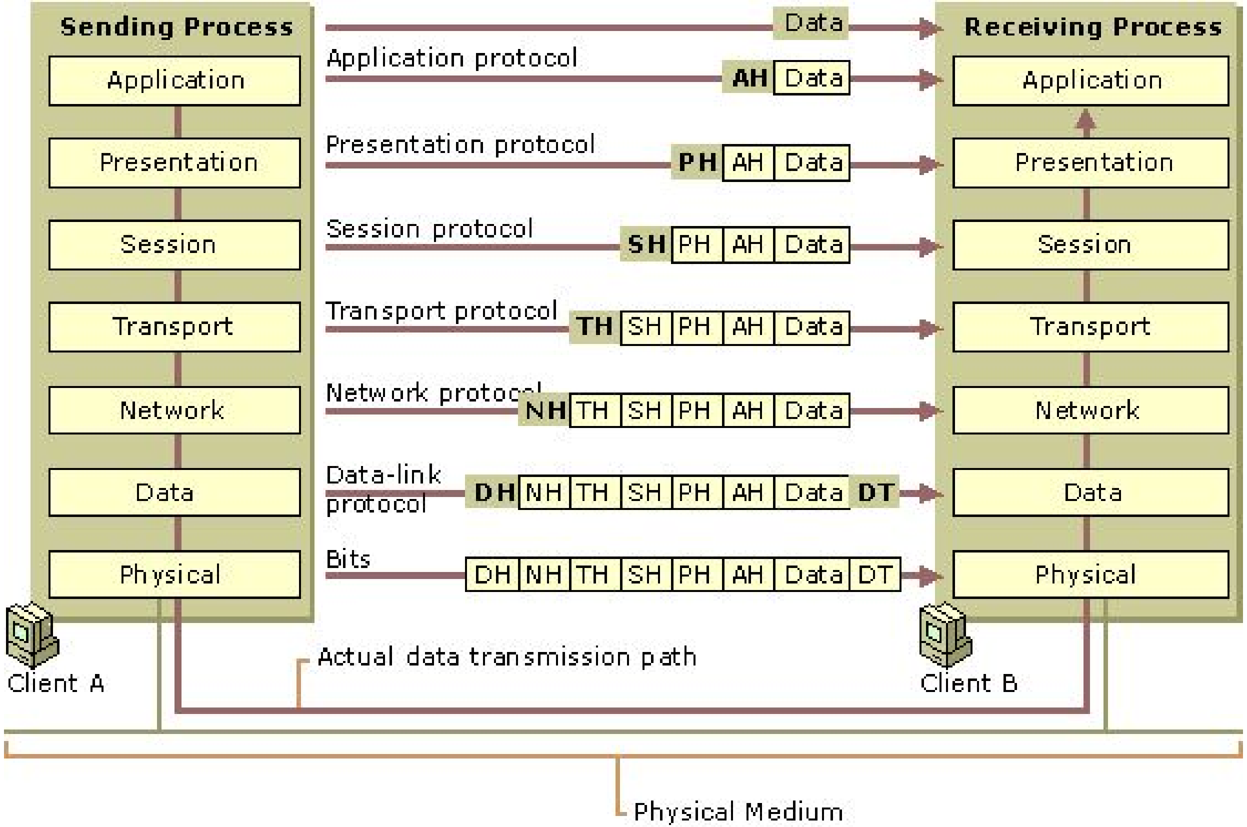
There's a stack! (5 Layer)



Encapsulation



Encapsulation

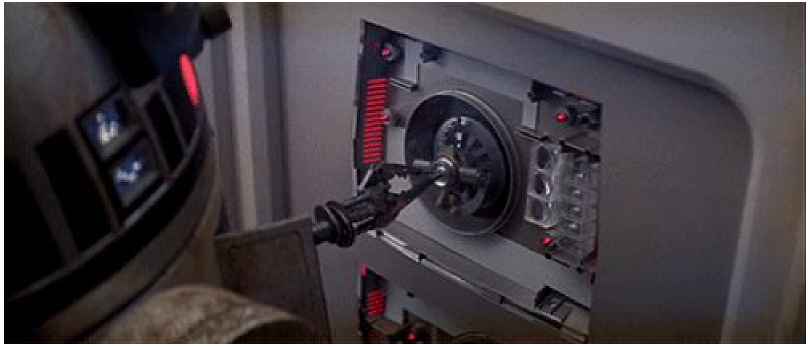


Application Layer

- What most people interface with
- Process-to-process communication within a network
  - local host, or remote host
- Relies on layers below for reliable/un-reliable “pipes” to other processes
- Protocols on protocols:
  - SMTP, IMAP, POP
  - HTTP, HTTPS
  - DNS
  - SSH
  - FTP
  - RDP
  - VNC
  - And many many more... But we stop at 256

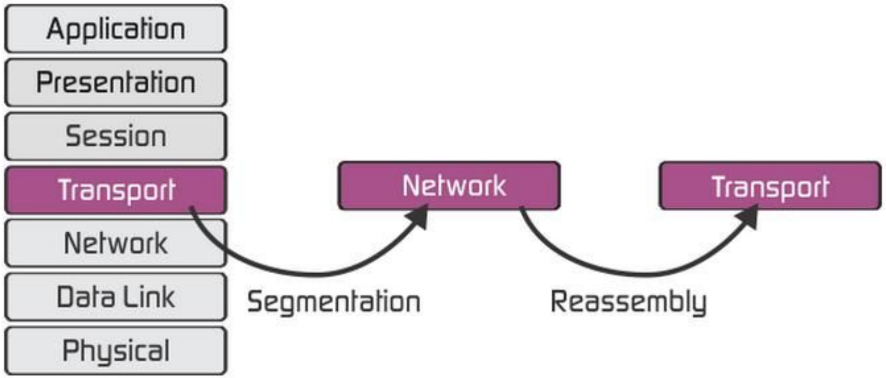
Application Layer

- Sockets
  - The “tube”
  - Endpoint of communication link exposed to application
- Example code
  - https://github.com/gwAdvNet2015/adv-net-samples/blob/master/sockets/server-tcp.c
  - https://github.com/gwAdvNet2015/adv-net-samples/blob/master/sockets/client-tcp.c



Transport Layer

- End-to-end communication services for applications
- Provides networking with
  - Reliability
  - Flow control
  - Connection-oriented data streaming
  - Multiplexing
- Protocols:
  - TCP
  - UDP
  - DCCP
  - RSVP
  - And some more



TCP vs UDP

TCP

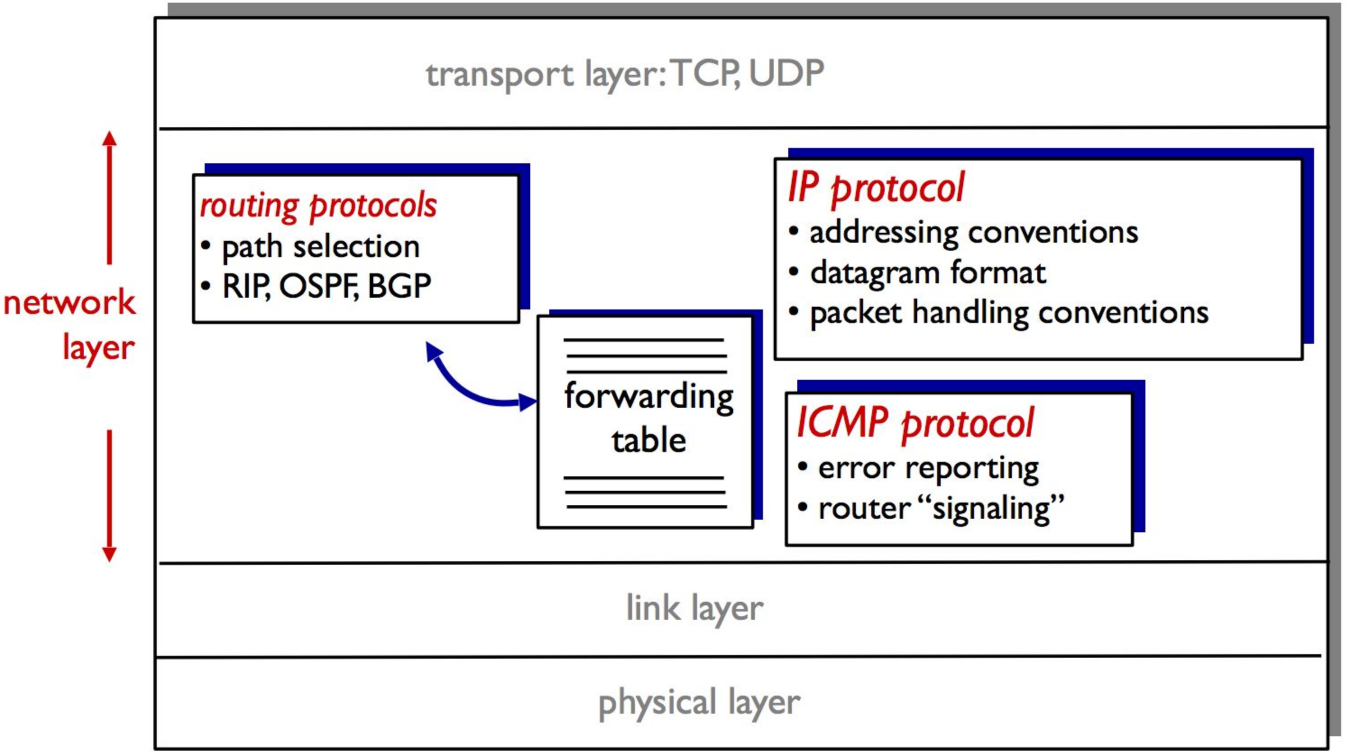
- Guaranteed protocol
    - What does this mean?
- Slower
- Acknowledged

UDP

- Not guaranteed protocol
    - What does this mean?
- Faster
- Non-acknowledged

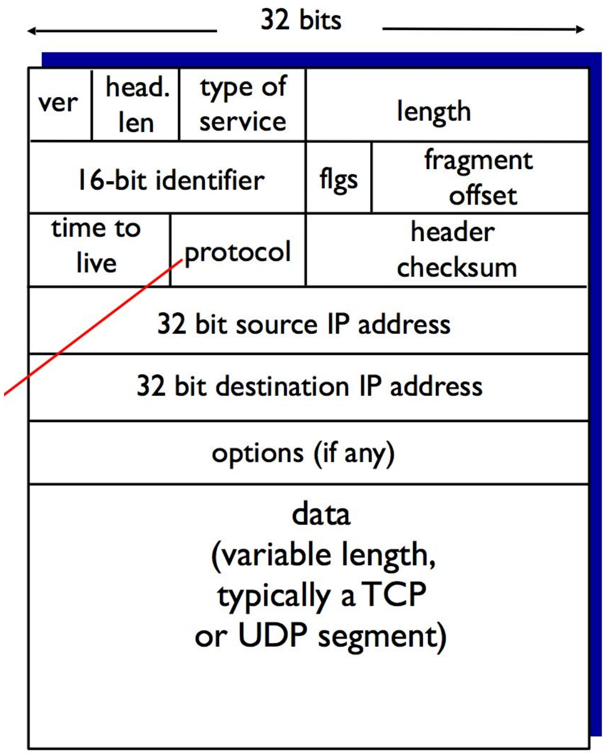
- Overheads?
- When to use TCP?
- When to use UDP?
- Which one needs connection setup?

Network Layer



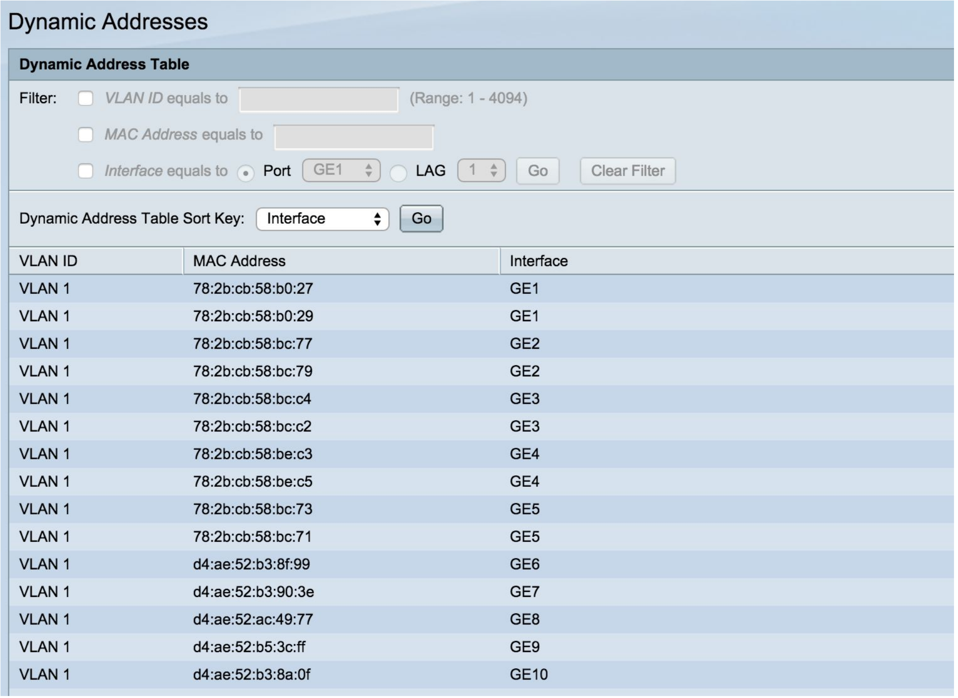
Network Layer

- IP Addresses come into play here
  - Host addressing
  - Message forwarding
  - No acknowledgement
- Various protocols such as IPv4/IPv6, ICMP, IPsec, IGMP, etc...



Link Layer

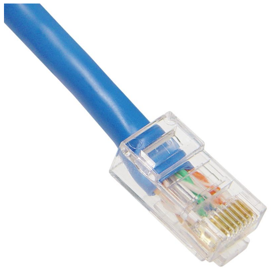
- Local networking
    - On the same LAN
- Routing using mac addresses
    - L2 Switches (yay for Cisco)
    - ARP Tables
- QoS control



VLAN ID	MAC Address	Interface
VLAN 1	78:2b:cb:58:b0:27	GE1
VLAN 1	78:2b:cb:58:b0:29	GE1
VLAN 1	78:2b:cb:58:bc:77	GE2
VLAN 1	78:2b:cb:58:bc:79	GE2
VLAN 1	78:2b:cb:58:bc:c4	GE3
VLAN 1	78:2b:cb:58:bc:c2	GE3
VLAN 1	78:2b:cb:58:be:c3	GE4
VLAN 1	78:2b:cb:58:be:c5	GE4
VLAN 1	78:2b:cb:58:bc:73	GE5
VLAN 1	78:2b:cb:58:bc:71	GE5
VLAN 1	d4:ae:52:b3:8f:99	GE6
VLAN 1	d4:ae:52:b3:90:3e	GE7
VLAN 1	d4:ae:52:ac:49:77	GE8
VLAN 1	d4:ae:52:b5:3c:ff	GE9
VLAN 1	d4:ae:52:b3:8a:0f	GE10

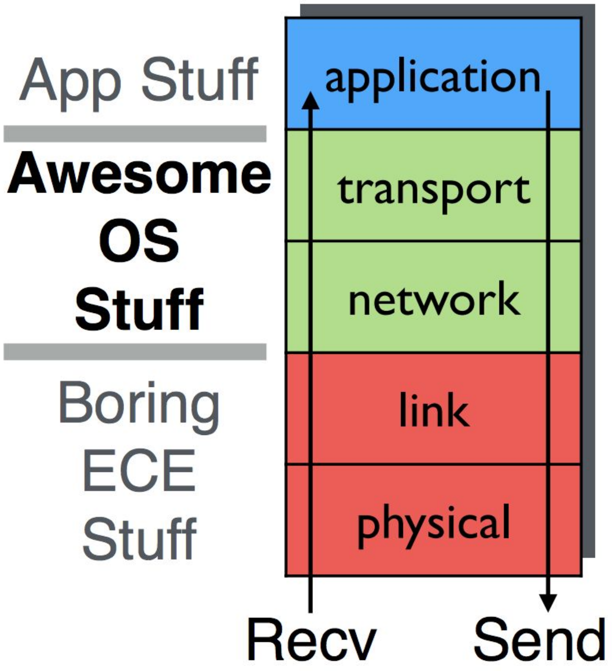
Physical Layer

- Analog data
  - Electrons and signals
- Physical medium for data to move on
  - Ethernet
  - Fiber
  - Copper
  - WiFi
- Dictates network topologies



RX/TX Process

- Why layers?
- Why encapsulation?
- How does the kernel handle packets at each layer?



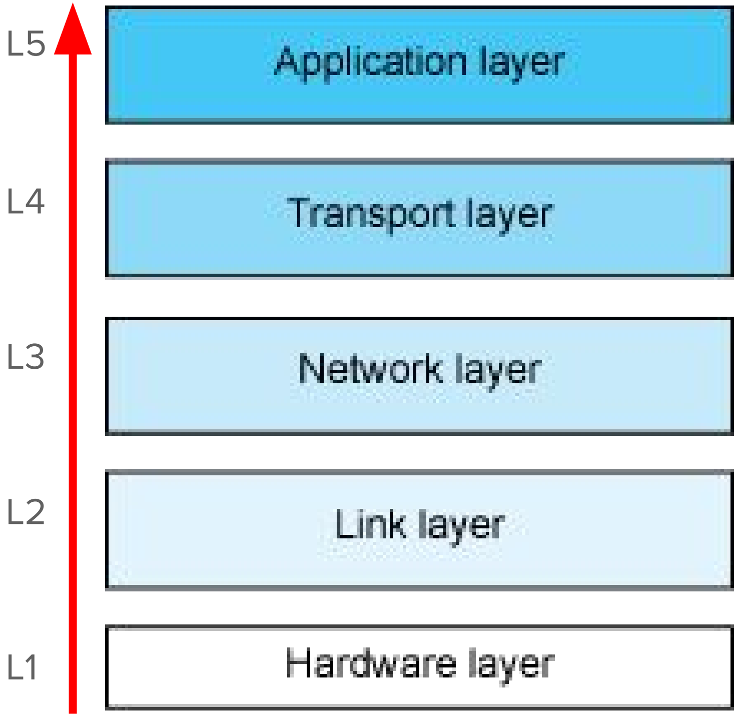
Go here

http://www.makelinux.net/kernel_map/

Receive packet

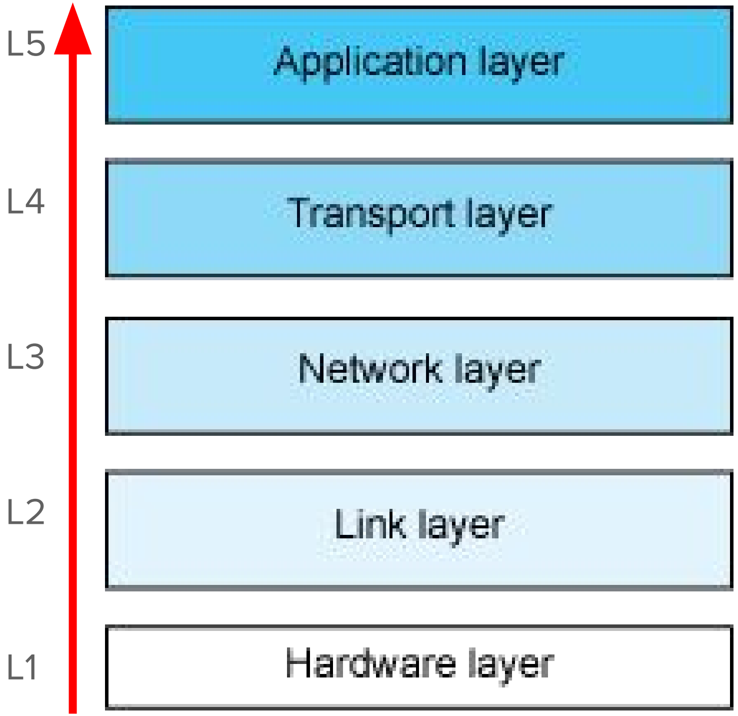
Go up the stack! (L1 -> L5/L7):

- NIC registers interrupt handler (L1/L2)
  - Packet copied into kernel memory
    - sk_buffer data struct to refer to it
- Calls ip_recv() (L2/L3)
  - http://lxr.free-electrons.com/source/net/ipv4/ip_input.c#L377
  - Processes IP header of packet
    - Local or to forward?
- Determine TCP vs UDP (L4)



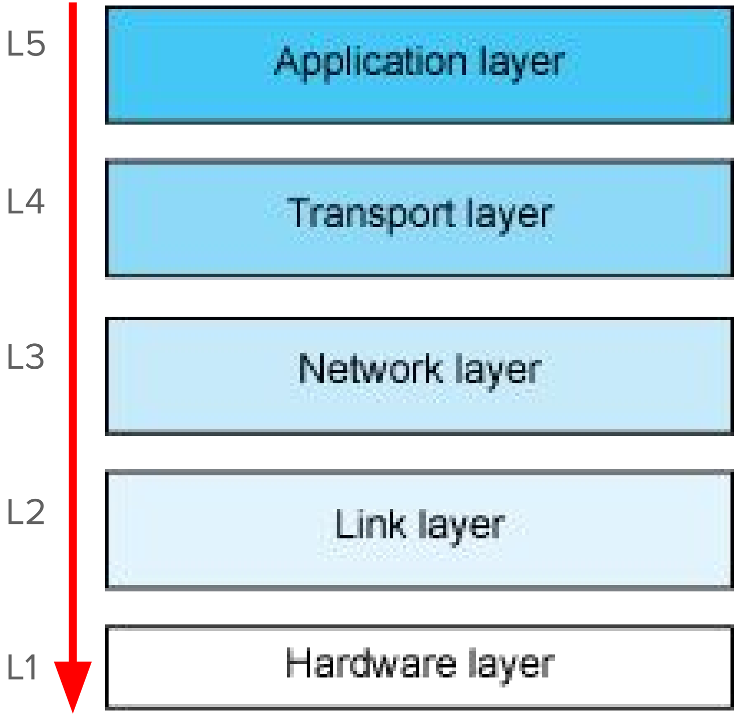
Receive packet

- Match packet to correct socket (L4/L5)
  - Each application that needs to network has a socket
  - Sockets identified with dst/src address and port
  - Ports allow kernel to segregate network traffic to specific sockets
- Wait for user to request...
- User calls recv() (L5)
  - http://linux.die.net/man/2/recv
- Now the application can do “stuffs” with the packet data



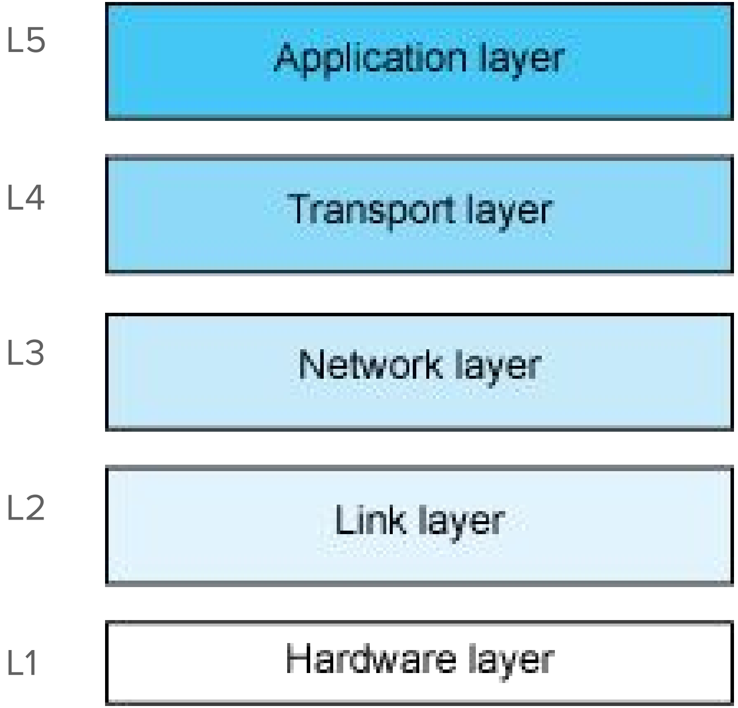
Transmit packet

- Do all that we just did, but backwards!
- Create a frame and encapsulate it with data needed for each layer (step down the stack)
- Initiated when user calls send()
  - http://linux.die.net/man/2/send
- And we just fall down the stack and out of the nic
  - Yes, gravity does this

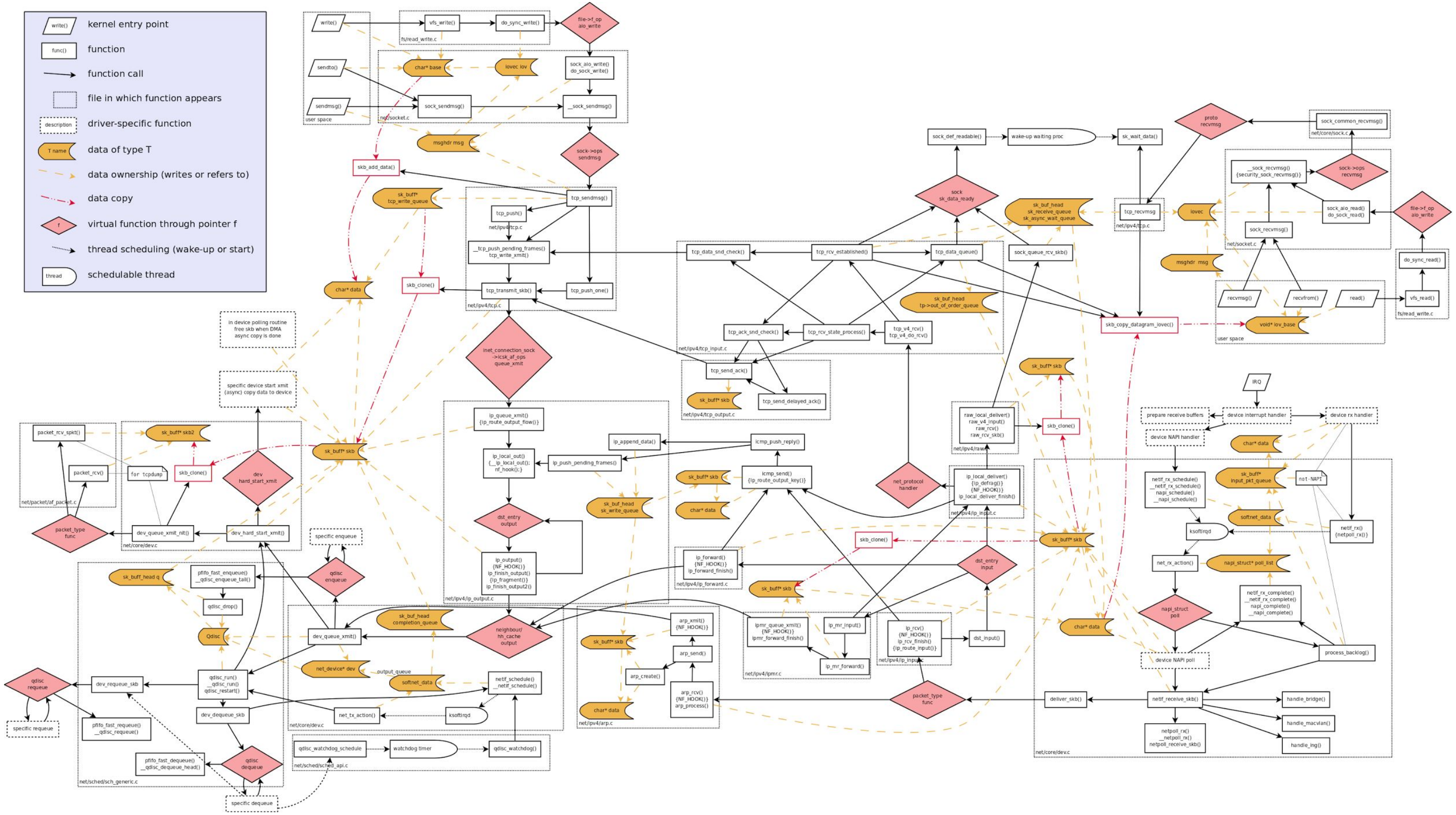


Transmit packet

- Do all that we just did, but backwards!
- Create a frame and encapsulate it with data needed for each layer (step down the stack)
- User calls send()
  - http://linux.die.net/man/2/send
- And we just fall down the stack and out of the nic
  - ~~Yes, gravity does this~~
  - Functions at each layer call the next



Network data flow through kernel



struct sk_buff

- Large struct which contains **control information** for the packet
- Fields that point to the L2, L3, and L4 layers are in the struct
- **Minimizes** copying overheads as packet moves through kernel
- Organized as a doubly linked list (struct sk_buff_head)
  - Example of RX/TX queues in sock.h
- Each layer of the network stack gets its own sk_buff
  - It is copied
  - Low overhead since most fields are unused or are pointers

L2 In Kernel

- Hardware pre-allocates sk_buffs
- These addresses are configured for DMA
    - Direct Memory Access is what allows hardware to access regions of memory without the CPU
- Interrupts
    - Slow
    - Every packet = new interrupt
    - All the packets??
- Polling
    - Kernel asks NIC for data
    - Slow poll rate?
- NAPI
    - Interrupts by default **but** when there’s high network traffic, kernel **switches** to polling

L3 In Kernel

- ARP
  - arp_rcv() function to handle arp requests and process
- IPv4
  - ip_rcv() function to handle IPv4
  - If the packet is destined for this machine, call
    - ip_local_deliver()
  - If the packet is destined for an address that is in the routing table
    - ip_forward()
    - Packet does not go to next layer, L4, if it is not destined for this machine
  - Calls raw_local_deliver() first before calling the L4 protocol handler in the sk_buff

L4 In Kernel

- Function tcp_v4_rcv() handles processing of TCP sequence
    - If there is an acknowledgement, further packets arrive
- Function udp_rcv() handles processing of UDP packets
- If user does not have a process accepting incoming packets, it is copied to the socket's queue
- If a user does have a process accepting incoming packets, then it is immediately copied to the socket

Networking

Layer
Application (7)
Presentation (6)
Session (5)
Transport (4)
Network (3)
Data Link (2)
Physical (1)

RX

TX

Further Reading

- https://en.wikipedia.org/wiki/OSI_model
- http://www.cs.unh.edu/cnrg/people/gherrin/linux-net.html
- https://thesquareplanet.com/blog/how-the-internet-works/
- https://github.com/gwAdvNet2015/adv-net-samples/wiki/slides/lec-4-network-os.pdf
- http://www.haifux.org/lectures/172/netLec.pdf
- http://www.haifux.org/lectures/180/netLec2.pdf
- http://www.haifux.org/lectures/219/netLec6.pdf
- http://www.linuxfoundation.org/collaborate/workgroups/networking/sk_buff
  - Must read
- http://www.linuxfoundation.org/collaborate/workgroups/networking/kernel_flow
  - Must read
- http://www.linuxfoundation.org/collaborate/workgroups/networking/socket_locks