# ECE560
# Computer and Information Security

# Fall 2023

## Networking Overview

Tyler Bletsch

Duke University

Some slides adapted from Brian Rogers (Duke)

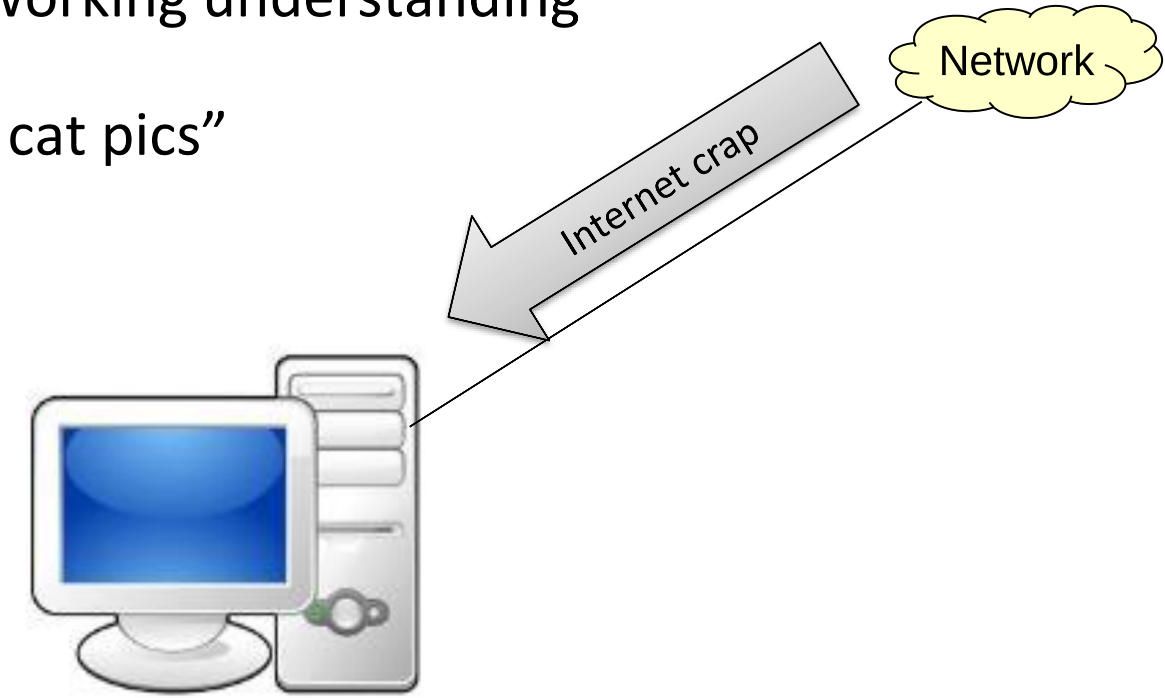
# Network fundamentals

This course isn’t a networking course, so we’ll just hit the highlights

- We want to hook computers together
- There’s a near-infinite number of ways to do this
- We’ll skip the theory and show you the common case

- Average person’s networking understanding

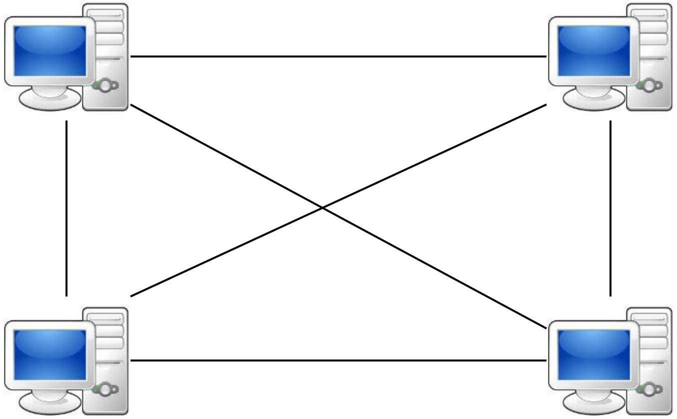
  “Gateway where I get cat pics”



# Network organization

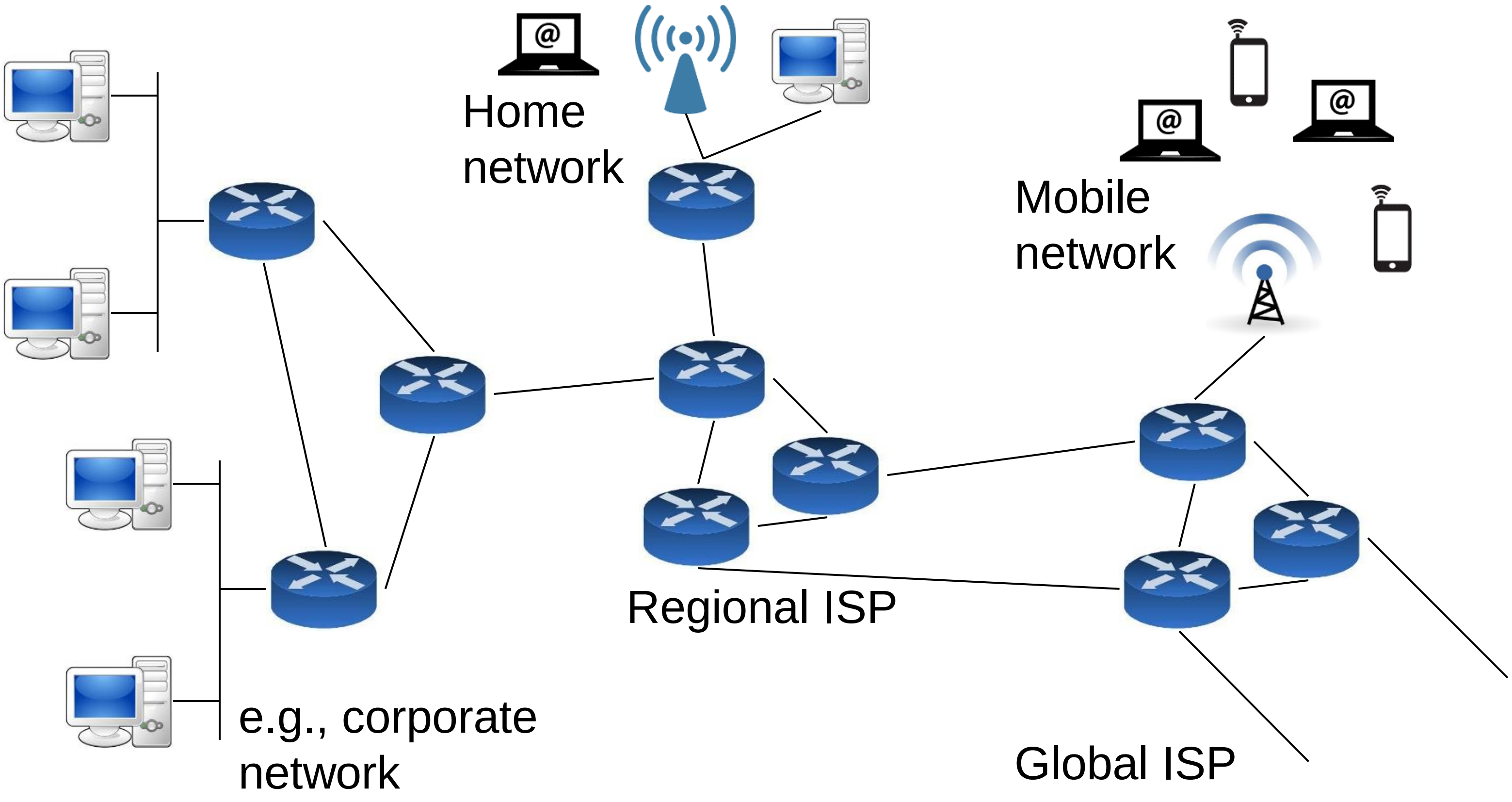
# Connectivity in the Internet

- A point-to-point mesh?
- Clearly not sustainable for large networks
  - $N^2$ links required
  - Add new endpoint: new link added to all existing endpoints



# Network Structure

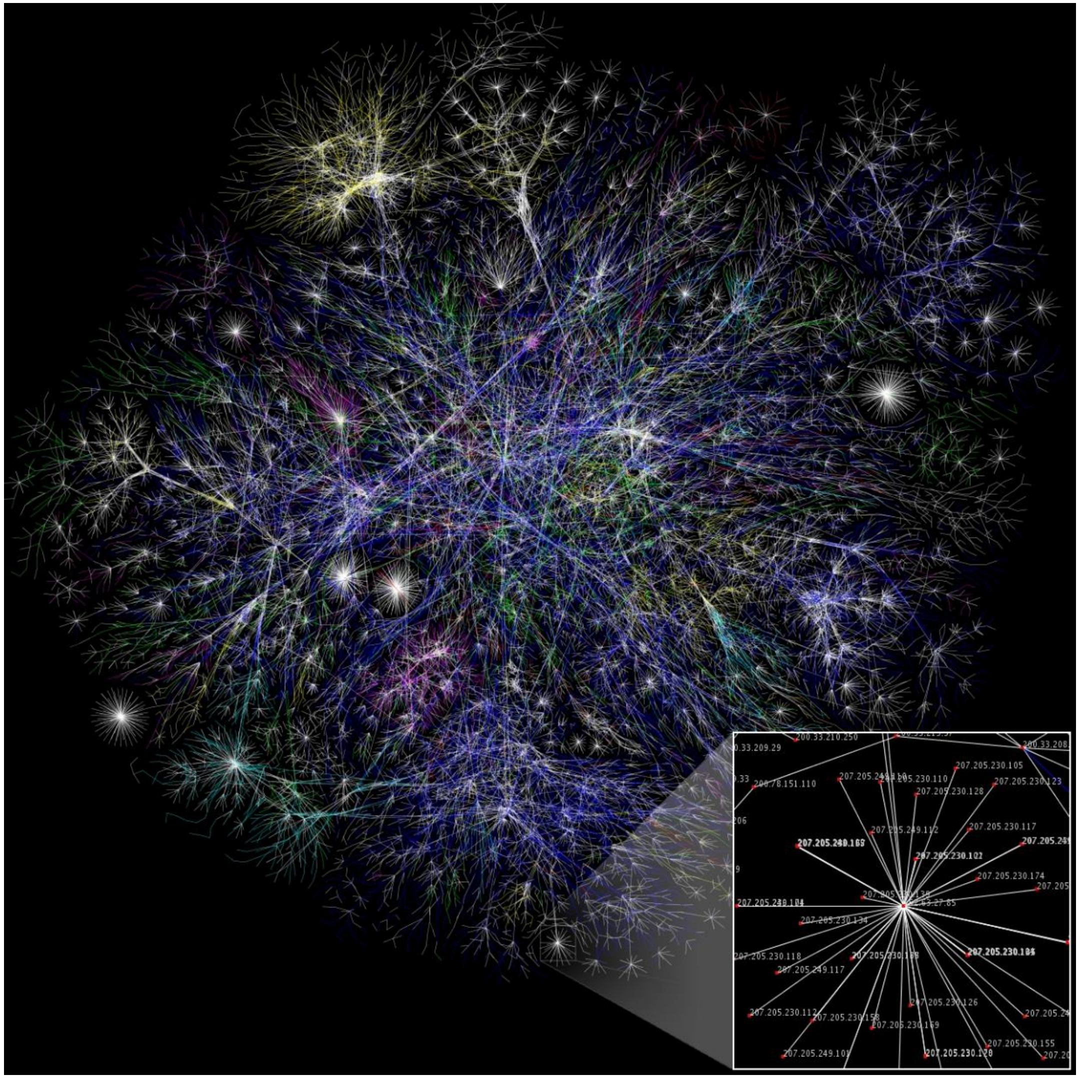
- Need to share infrastructure!
- Routers and switches (intermediate nodes) allow sharing



# Internet Backbone

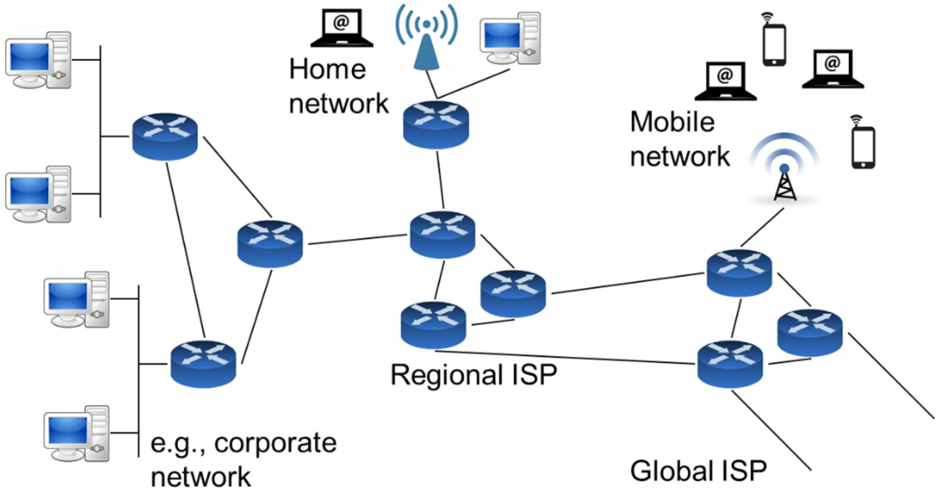
- From Wikipedia:

- Due to sharing, we get a structure that looks like this
- Localized "stars" connected to others

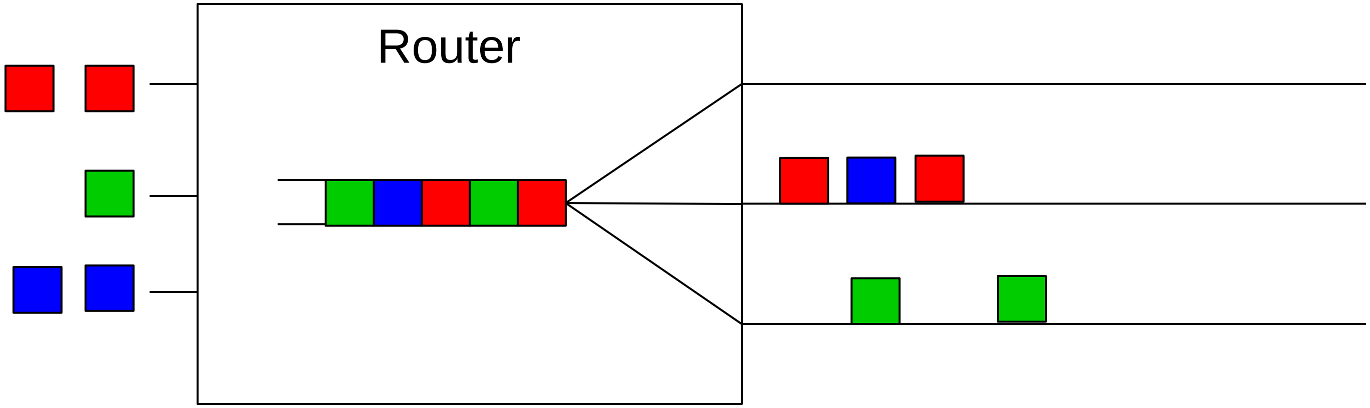


# Usage Models

- Network endpoints run application programs
  - Web browser, email client, ssh, etc.
- Client / Server model
  - Client endpoints requests a service from a server
  - E.g. client / server web page service
- Peer-to-peer (P2P)
  - Direct client communication (e.g. Skype, BitTorrent)



# Packet Switched Routers



- Multiplex w/ queue(s) in the router
- Demultiplex with packet header info:
  - Destination endpoint

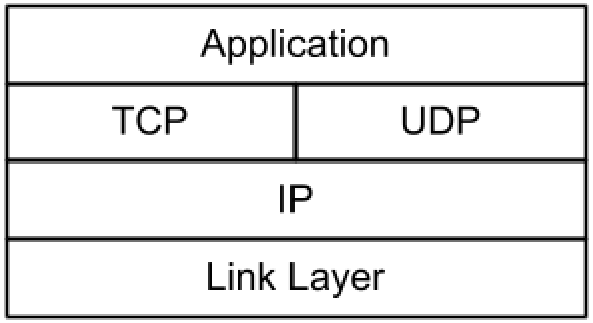
# Managing Complexity

- Very large number of computers
- Incredible variety of technologies
    - Each with very different constraints
- No single administrative entity
- Evolving demands, protocols, applications
    - Each with very different requirements!
- How do we make sense of all this?

# Networking layers

# Layering

- We see layers of abstraction



Application

TCP | UDP

IP

Link Layer

- Separation of concerns
  - Break problem into separate parts
  - Solve each one independently
  - Tie together through common interfaces: abstraction
  - Encapsulate data from layer above inside data from layer below
  - Allow independent evolution

# Layering

- We see layers f abstraction

Applicat

- Separat
  - B
  - Solve
  - Tie toget
  - Encapsu ye below
  - Allow

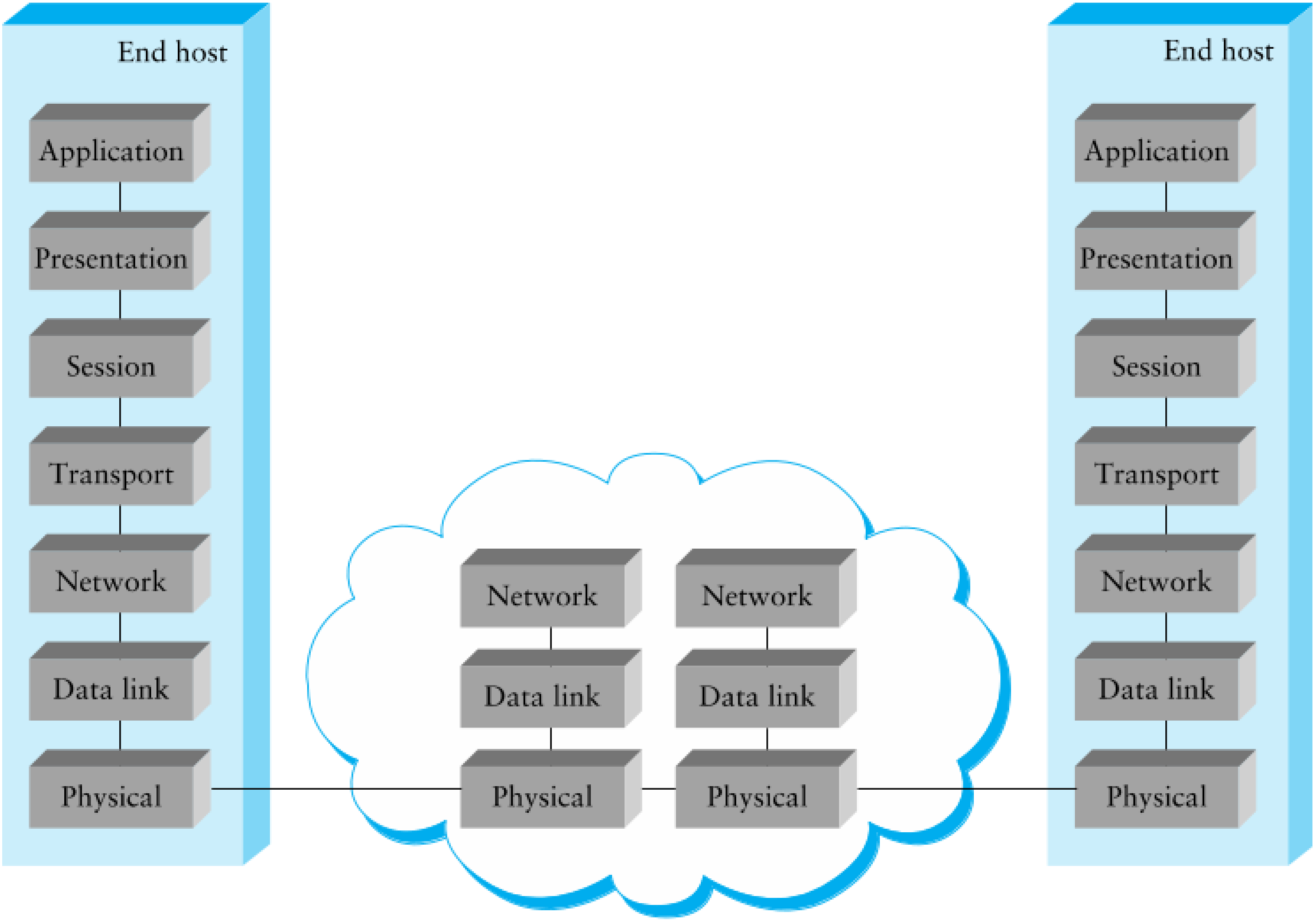
# Layering done wrong invites security vulnerabilities!

- Layering is a form of modularity; modularity is good
  ***IF and ONLY IF***
  you don't make any dangerous assumptions!

- Networking stack is good, common, but has had lots of vulnerabilities
  - Many vulnerabilities of the form "layer X makes an implicit assumption about data from layer Y"
  - Example: Receiving a packet with an Ethernet frame size in conflict with TCP packet size -> Buggy network code segfaults

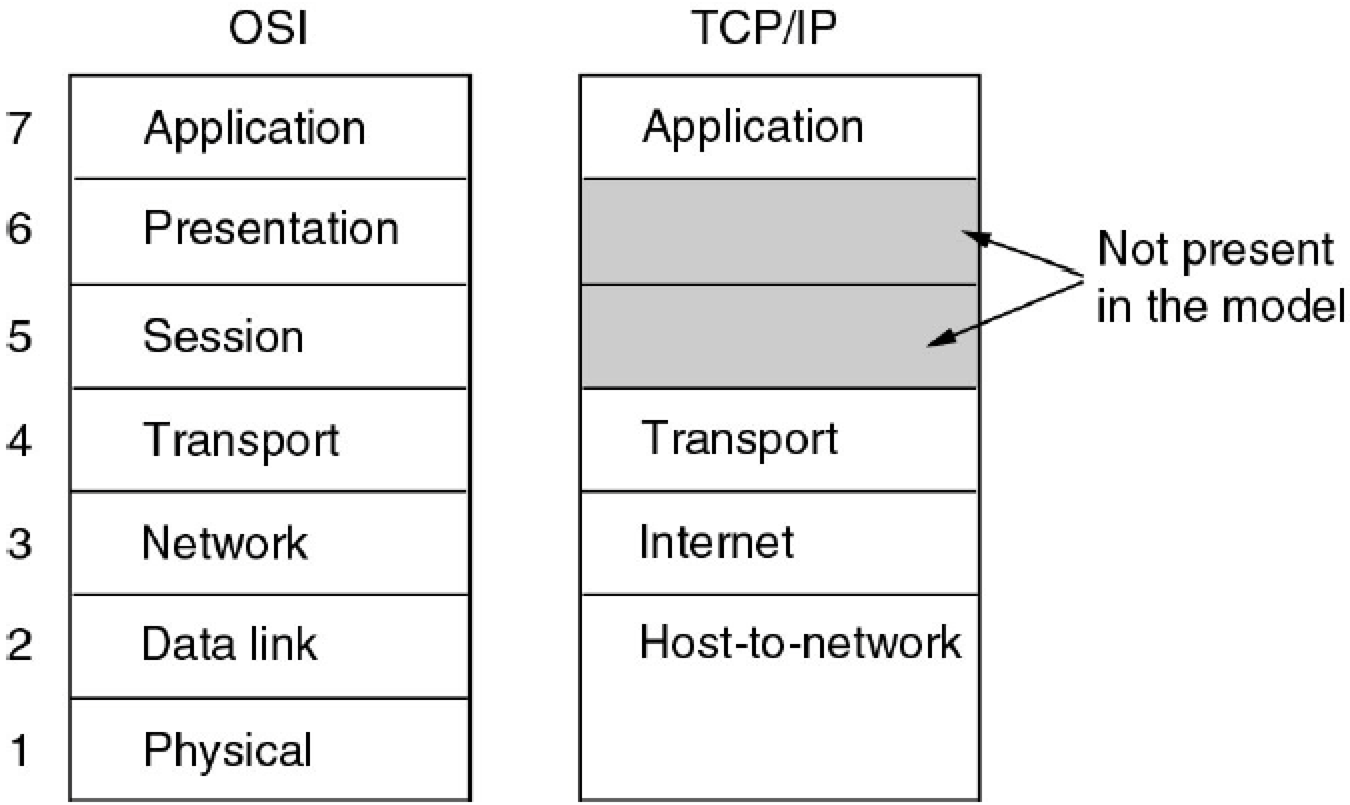
- **Rule of thumb**:
  *Be strict in what you send and check carefully what you receive*

# OSI Reference Model



One or more nodes
within the network

# TCP/IP Model



| | OSI | TCP/IP |
|---|---|---|
| 7 | Application | Application |
| 6 | Presentation | |
| 5 | Session | |
| 4 | Transport | Transport |
| 3 | Network | Internet |
| 2 | Data link | Host-to-network |
| 1 | Physical | |

Not present in the model

# Layer 1 & 2

- Layer 1: Physical Layer
  - Encoding of bits to send over a single physical link

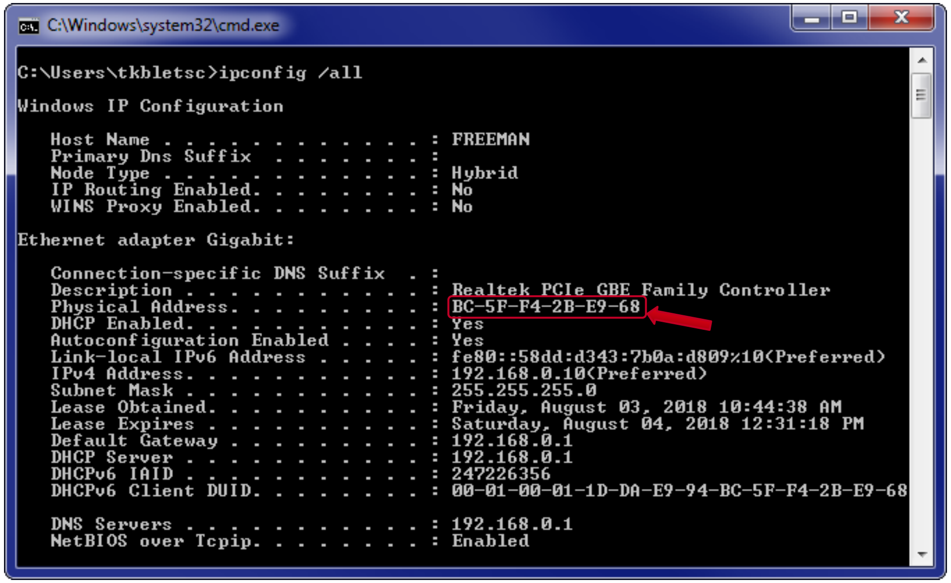
Examples: Ethernet, 802.11 WiFi
(the part of the spec that says how to send bits)

- Layer 2: Link Layer
  - Framing and transmission of a collection of bits into individual messages sent across a single subnetwork (one physical topology)
  - Provides local addressing (MAC)
  - May involve multiple physical links
  - Often the technology supports broadcast: every “node” connected to the subnet receives

Examples: Ethernet, 802.11 WiFi
(the part of the spec that how to send packets to a host on this network)

# Ethernet/WiFi and MAC addresses

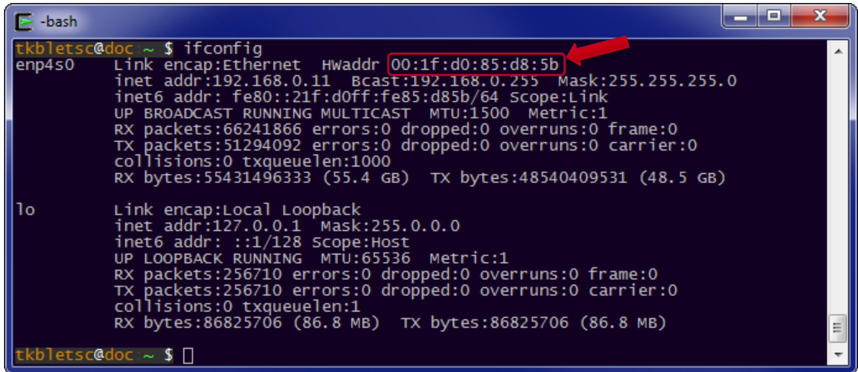
- Each network interface has a **MAC address** ("Media Access Control"): a 48-bit value burned into network card; globally unique
  - First 3 bytes tell the manufacturer (OUI: Organizationally Unique Identifier)
  - Last 3 bytes are made to be unique by that manufacturer
- Usually written as colon-delimited hex: BC:5F:F4:2B:E9:68
- Only meaningful on a single **local area network** (wired or wireless)
- Not transmitted across internet



```
C:\Users\tkbletsc>ipconfig /all

Windows IP Configuration

   Host Name . . . . . . . . . . . . : FREEMAN
   Primary Dns Suffix  . . . . . . . :
   Node Type . . . . . . . . . . . . : Hybrid
   IP Routing Enabled. . . . . . . . : No
   WINS Proxy Enabled. . . . . . . . : No

Ethernet adapter Gigabit:

   Connection-specific DNS Suffix  . :
   Description . . . . . . . . . . . : Realtek PCIe GBE Family Controller
   Physical Address. . . . . . . . . : BC-5F-F4-2B-E9-68
   DHCP Enabled. . . . . . . . . . . : Yes
   Autoconfiguration Enabled . . . . : Yes
   Link-local IPv6 Address . . . . . : fe80::58dd:d343:7b0a:d809%10(Preferred)
   IPv4 Address. . . . . . . . . . . : 192.168.0.10(Preferred)
   Subnet Mask . . . . . . . . . . . : 255.255.255.0
   Lease Obtained. . . . . . . . . . : Friday, August 03, 2018 10:44:38 AM
   Lease Expires . . . . . . . . . . : Saturday, August 04, 2018 12:31:18 PM
   Default Gateway . . . . . . . . . : 192.168.0.1
   DHCP Server . . . . . . . . . . . : 192.168.0.1
   DHCPv6 IAID . . . . . . . . . . . : 247226356
   DHCPv6 Client DUID. . . . . . . . : 00-01-00-01-1D-DA-E9-94-BC-5F-F4-2B-E9-68

   DNS Servers . . . . . . . . . . . : 192.168.0.1
   NetBIOS over Tcpip. . . . . . . . : Enabled
```

Windows

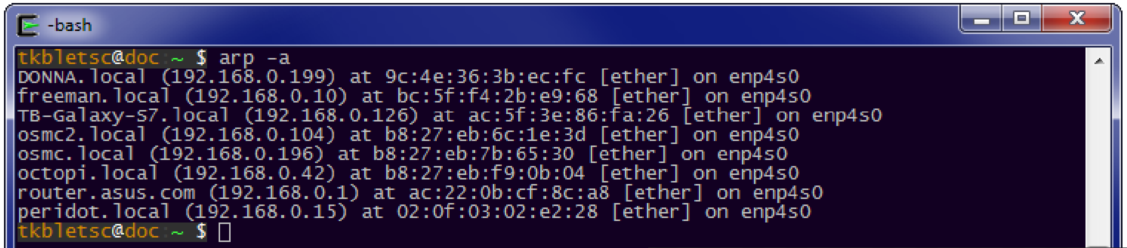
Linux



```
tkbletsc@doc ~ $ ifconfig
enp4s0    Link encap:Ethernet  HWaddr 00:1f:d0:85:d8:5b
          inet addr:192.168.0.11  Bcast:192.168.0.255  Mask:255.255.255.0
          inet6 addr: fe80::21f:d0ff:fe85:d85b/64 Scope:Link
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:66241866 errors:0 dropped:0 overruns:0 frame:0
          TX packets:51294092 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:55431496333 (55.4 GB)  TX bytes:48540409531 (48.5 GB)

lo        Link encap:Local Loopback
          inet addr:127.0.0.1  Mask:255.0.0.0
          inet6 addr: ::1/128 Scope:Host
          UP LOOPBACK RUNNING  MTU:65536  Metric:1
          RX packets:256710 errors:0 dropped:0 overruns:0 frame:0
          TX packets:256710 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1
          RX bytes:86825706 (86.8 MB)  TX bytes:86825706 (86.8 MB)

tkbletsc@doc ~ $ 
```

# Layer 1/2 demo: ARP

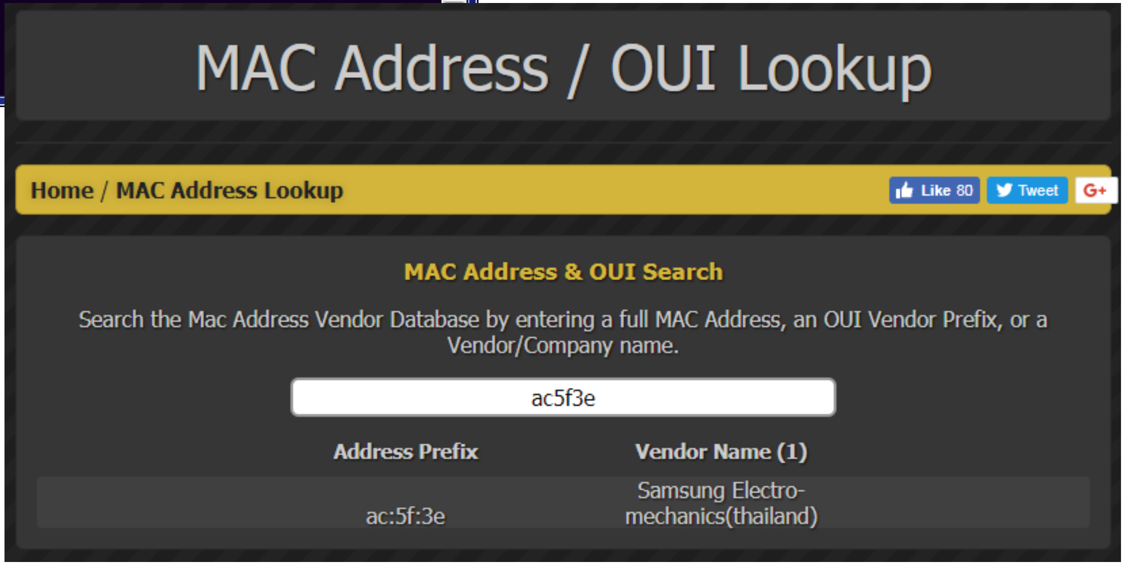
- Address Resolution Protocol (ARP): how we figure out the layer 2 address (MAC address) for a given layer 3 address (IP address)
  - Can inquire to see known MAC addresses
  - Can use OUI (first 3 bytes) to check manufacturer of devices!



```
tkbletsc@doc ~ $ arp -a
DONNA.local (192.168.0.199) at 9c:4e:36:3b:ec:fc [ether] on enp4s0
freeman.local (192.168.0.10) at bc:5f:f4:2b:e9:68 [ether] on enp4s0
TB-Galaxy-S7.local (192.168.0.126) at ac:5f:3e:86:fa:26 [ether] on enp4s0
osmc2.local (192.168.0.104) at b8:27:eb:6c:1e:3d [ether] on enp4s0
osmc.local (192.168.0.196) at b8:27:eb:7b:65:30 [ether] on enp4s0
octopi.local (192.168.0.42) at b8:27:eb:f9:0b:04 [ether] on enp4s0
router.asus.com (192.168.0.1) at ac:22:0b:cf:8c:a8 [ether] on enp4s0
peridot.local (192.168.0.15) at 02:0f:03:02:e2:28 [ether] on enp4s0
tkbletsc@doc ~ $ 
```

**Left**: ARP listing for my home server

**Below**: Lookup of manufacturer of the “TB-Galaxy-S7” device



MAC Address / OUI Lookup

MAC Address & OUI Search

Search the Mac Address Vendor Database by entering a full MAC Address, an OUI Vendor Prefix, or a Vendor/Company name.

ac5f3e

| Address Prefix | Vendor Name (1) |
|---|---|
| ac:5f:3e | Samsung Electro-mechanics(thailand) |

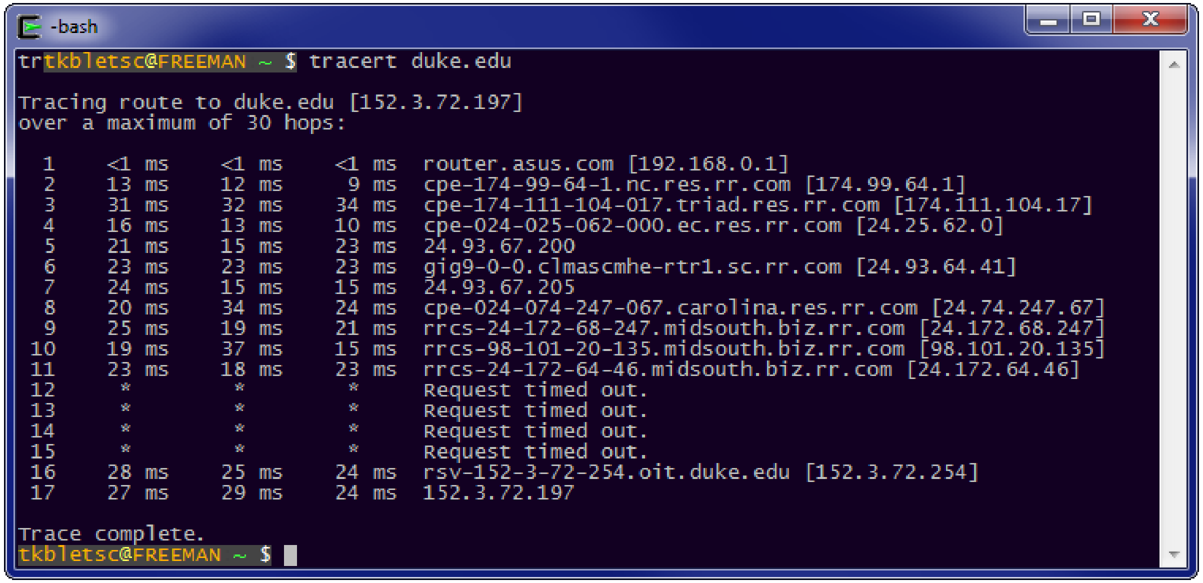
http://www.whatsmyip.org/mac-address-lookup/

# Layer 3

Example: Internet Protocol (IP)
(how to send packets between networks)

- Bridges multiple “subnets” to provide end-to-end connectivity between nodes
- Provides global addressing (IP addresses)
- Only provides best-effort delivery of data
  - No retransmissions, etc.
- Works across different link technologies

**Below:** Diagnostic tool showing the IP addresses passed on the way from my home to duke.edu



```
trtkbletsc@FREEMAN ~ $ tracert duke.edu

Tracing route to duke.edu [152.3.72.197]
over a maximum of 30 hops:

  1    <1 ms    <1 ms    <1 ms  router.asus.com [192.168.0.1]
  2    13 ms    12 ms     9 ms  cpe-174-99-64-1.nc.res.rr.com [174.99.64.1]
  3    31 ms    32 ms    34 ms  cpe-174-111-104-017.triad.res.rr.com [174.111.104.17]
  4    16 ms    13 ms    10 ms  cpe-024-025-062-000.ec.res.rr.com [24.25.62.0]
  5    21 ms    15 ms    23 ms  24.93.67.200
  6    23 ms    23 ms    23 ms  gig9-0-0.clmascmhe-rtr1.sc.rr.com [24.93.64.41]
  7    24 ms    15 ms    15 ms  24.93.67.205
  8    20 ms    34 ms    24 ms  cpe-024-074-247-067.carolina.res.rr.com [24.74.247.67]
  9    25 ms    19 ms    21 ms  rrcs-24-172-68-247.midsouth.biz.rr.com [24.172.68.247]
 10    19 ms    37 ms    15 ms  rrcs-98-101-20-135.midsouth.biz.rr.com [98.101.20.135]
 11    23 ms    18 ms    23 ms  rrcs-24-172-64-46.midsouth.biz.rr.com [24.172.64.46]
 12     *        *        *     Request timed out.
 13     *        *        *     Request timed out.
 14     *        *        *     Request timed out.
 15     *        *        *     Request timed out.
 16    28 ms    25 ms    24 ms  rsv-152-3-72-254.oit.duke.edu [152.3.72.254]
 17    27 ms    29 ms    24 ms  152.3.72.197

Trace complete.
tkbletsc@FREEMAN ~ $
```

# IPv4 addresses

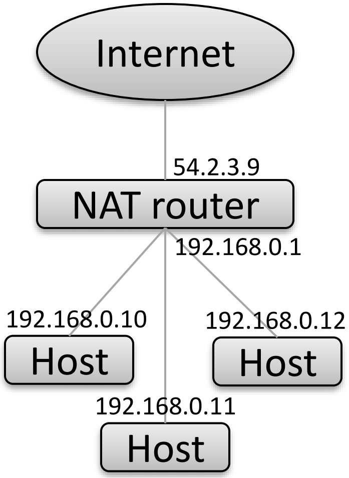
- IPv4 address is 32-bit address that is (*theroetically*) globally unique; identifies interface on the internet.
- Written as "dotted decimal" of the four bytes, e.g. "141.9.68.24".
  - So each number ("octet") can be 0-255.
- Subnets
  - An address can have its bits divided into <u>network</u> and <u>host</u>.
  - We describe a network in dotted decimal with a suffix saying how many bits are in the network part, e.g.: 181.41.0.0/18 – this is a **subnet**.
  - A mask of one bits covering the network portion is called the **netmask**; for 181.41.0.0/18, the netmask would be 255.255.192.0
  - The number of hosts that fit in a subnet is $2^{32-n} – 2$ (Minus two is because the all-zeroes host and all-ones host are special)
  - IP address assignment is hierarchical: Countries get IP ranges and assign to registrars who then divide them among customers (ISPs, companies, etc.).
  - The country of Aruba has 181.41.0.0/18 and a few others. For a long time, IBM had 9.0.0.0/8.

This tool will let you play with subnets

# Modern caveats (1)

- Some IP addresses are special:
  - **Loopback**: 127.0.0.1 always refers the machine you're on (actually it's all of 127.0.0.0/8)
  - **Private**: 10.0.0.0/8, 172.16.0.0/12, 192.168.0.0/16 – not allowed on internet
  - **Link-local**: 169.254.0.0/16 – auto-assigned when no network services are up
  - Others (see IANA IPv4 Special-Purpose Address Registry)

- We're running out of 32-bit IP addresses, so **NAT (Network Address Translation)** was invented
  - Have just one "real" public IP address at network boundary, assign private IP addresses internally and translate at border
  - Extremely common – real direct internet connections are rare (this is good, as NAT doubles as a firewall)



# Modern caveats (2)

- IP-to-interface mapping is actually more flexible:
  - For performance/reliability, an IP may be span multiple interfaces
  - For manageability reasons, an interface may have >1 IP address

- All of the above refers to IP version 4 ("**IPv4**")
- **IPv6** is being deployed now (and has been for the past 20 years)
  - IPv6 addresses are 128 bits (16 bytes) instead of 32 bits (4 bytes)
  - Written as colon-delimited 16-bit hex words:

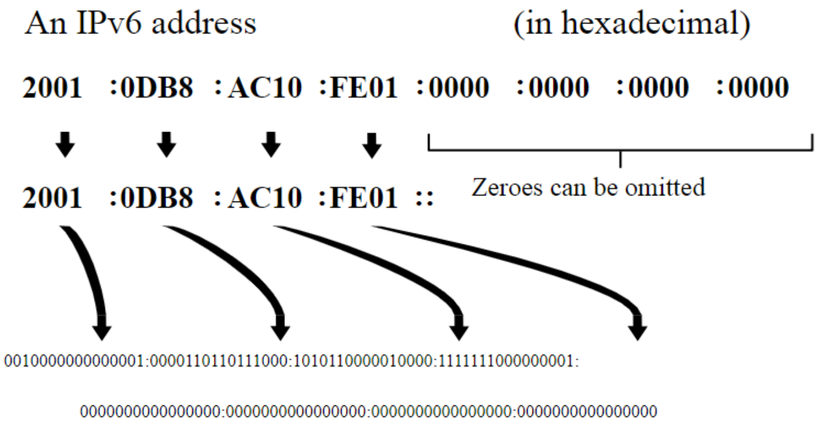
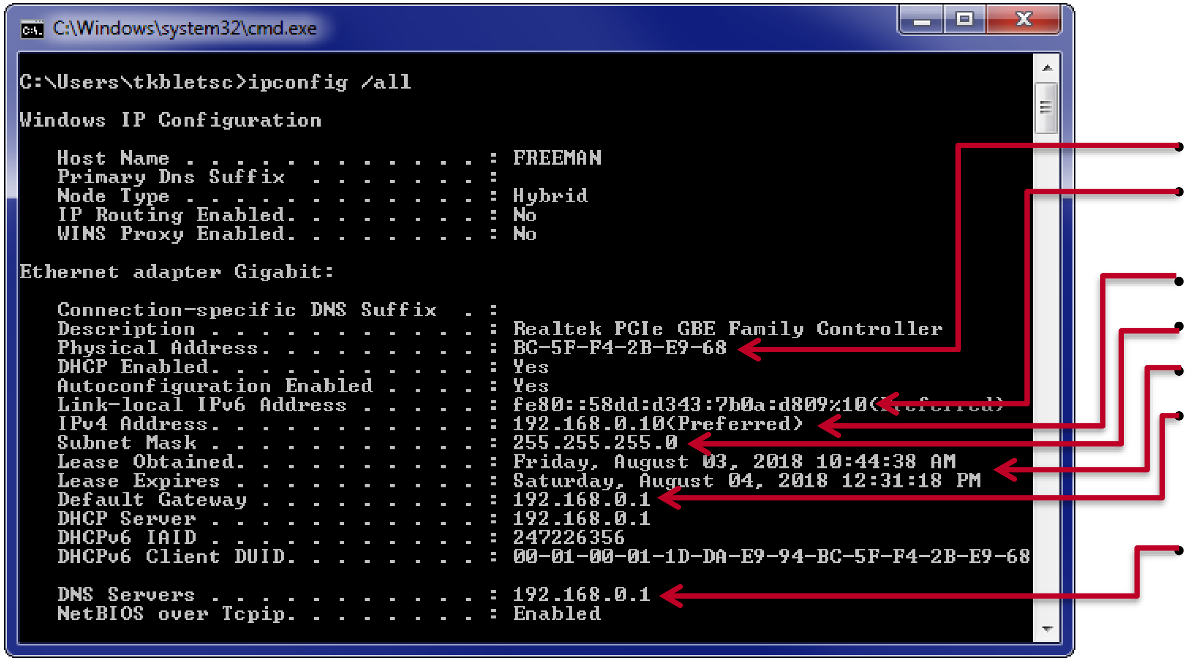


Figure from Wikipedia "IPv6"

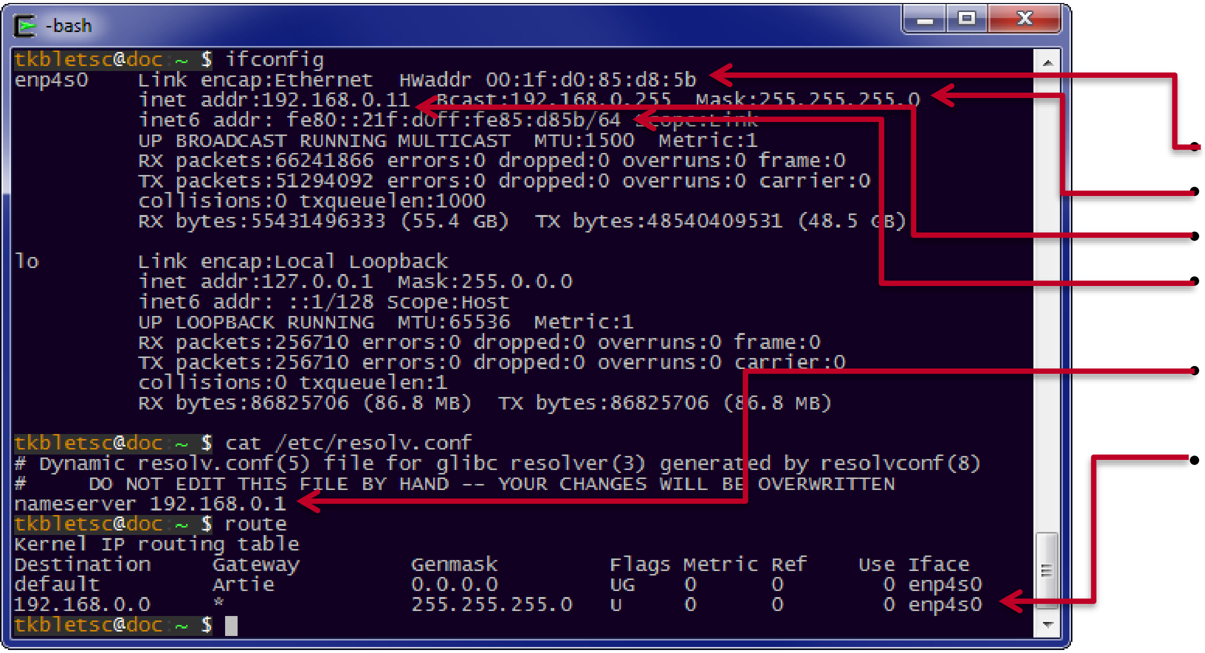
# Looking at real configs: Windows



```
C:\Users\tkbletsc>ipconfig /all

Windows IP Configuration

   Host Name . . . . . . . . . . . . : FREEMAN
   Primary Dns Suffix  . . . . . . . :
   Node Type . . . . . . . . . . . . : Hybrid
   IP Routing Enabled. . . . . . . . : No
   WINS Proxy Enabled. . . . . . . . : No

Ethernet adapter Gigabit:

   Connection-specific DNS Suffix  . :
   Description . . . . . . . . . . . : Realtek PCIe GBE Family Controller
   Physical Address. . . . . . . . . : BC-5F-F4-2B-E9-68
   DHCP Enabled. . . . . . . . . . . : Yes
   Autoconfiguration Enabled . . . . : Yes
   Link-local IPv6 Address . . . . . : fe80::58dd:d343:7b0a:d809%10(Preferred)
   IPv4 Address. . . . . . . . . . . : 192.168.0.10(Preferred)
   Subnet Mask . . . . . . . . . . . : 255.255.255.0
   Lease Obtained. . . . . . . . . . : Friday, August 03, 2018 10:44:38 AM
   Lease Expires . . . . . . . . . . : Saturday, August 04, 2018 12:31:18 PM
   Default Gateway . . . . . . . . . : 192.168.0.1
   DHCP Server . . . . . . . . . . . : 192.168.0.1
   DHCPv6 IAID . . . . . . . . . . . : 247226356
   DHCPv6 Client DUID. . . . . . . . : 00-01-00-01-1D-DA-E9-94-BC-5F-F4-2B-E9-68

   DNS Servers . . . . . . . . . . . : 192.168.0.1
   NetBIOS over Tcpip. . . . . . . . : Enabled
```

- MAC address
- IPv6 address (link local – not routed to internet in this config)
- IPv4 address (NAT routed private IP)
- Subnet mask (shows this is a /24 network)
- DHCP lease info
- Gateway: IP address we sent stuff to go get to the internet (NAT router in this case)
- DNS server: IP address we look up names with (my router does this too in this case)

# Looking at real configs: Linux



```
tkbletsc@doc ~ $ ifconfig
enp4s0    Link encap:Ethernet  HWaddr 00:1f:d0:85:d8:5b
          inet addr:192.168.0.11  Bcast:192.168.0.255  Mask:255.255.255.0
          inet6 addr: fe80::21f:d0ff:fe85:d85b/64 Scope:Link
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:66241866 errors:0 dropped:0 overruns:0 frame:0
          TX packets:51294092 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:55431496333 (55.4 GB)  TX bytes:48540409531 (48.5 GB)

lo        Link encap:Local Loopback
          inet addr:127.0.0.1  Mask:255.0.0.0
          inet6 addr: ::1/128 Scope:Host
          UP LOOPBACK RUNNING  MTU:65536  Metric:1
          RX packets:256710 errors:0 dropped:0 overruns:0 frame:0
          TX packets:256710 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1
          RX bytes:86825706 (86.8 MB)  TX bytes:86825706 (86.8 MB)

tkbletsc@doc ~ $ cat /etc/resolv.conf
# Dynamic resolv.conf(5) file for glibc resolver(3) generated by resolvconf(8)
#     DO NOT EDIT THIS FILE BY HAND -- YOUR CHANGES WILL BE OVERWRITTEN
nameserver 192.168.0.1
tkbletsc@doc ~ $ route
Kernel IP routing table
Destination     Gateway         Genmask         Flags Metric Ref    Use Iface
default         Artie           0.0.0.0         UG    0      0        0 enp4s0
192.168.0.0     *               255.255.255.0   U     0      0        0 enp4s0
tkbletsc@doc ~ $
```

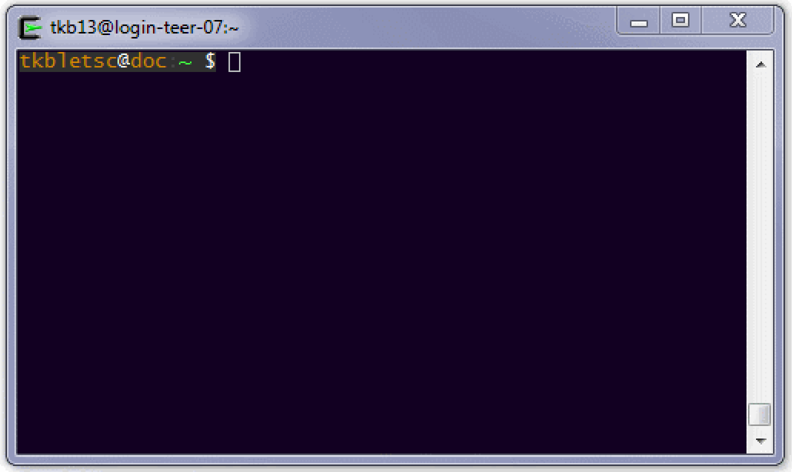
- MAC address
- Subnet mask (show this is a /24 network)
- IPv4 address (NAT routed private IP)
- IPv6 address (link local – not routed to internet in this config)
- DNS server: IP address we look up names with (my router does this too in this case)
- Gateway: IP address we sent stuff to go get to the internet (NAT router in this case)

# Layer 4

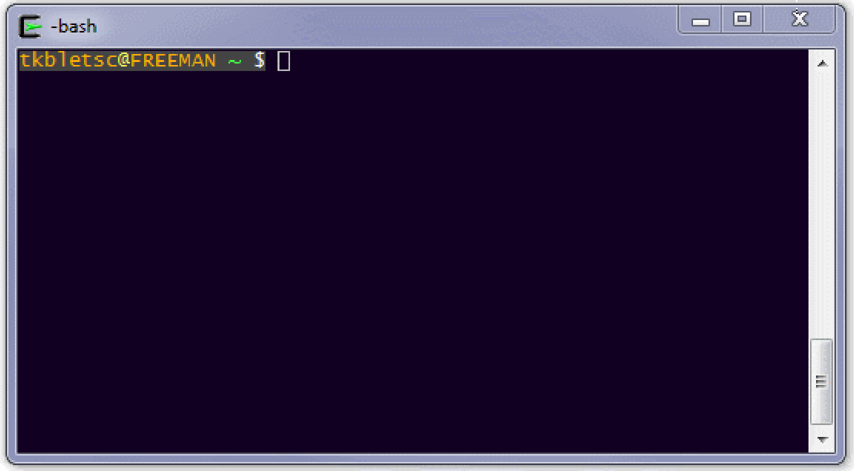
Example: TCP/UDP
(how to establish a logical channel, maybe even a reliable channel)

- End-to-end communication between processes
- Different types of services provided:
  - UDP: unreliable datagrams
  - TCP: reliable byte stream
- “Reliable” = keeps track of what data were received properly and retransmits as necessary
- This is the layer that applications talk with

**Below:** Sending data between two computers via a raw TCP socket using the ‘netcat’ (nc) tool.



```
tkbletsc@doc ~ $ 
```



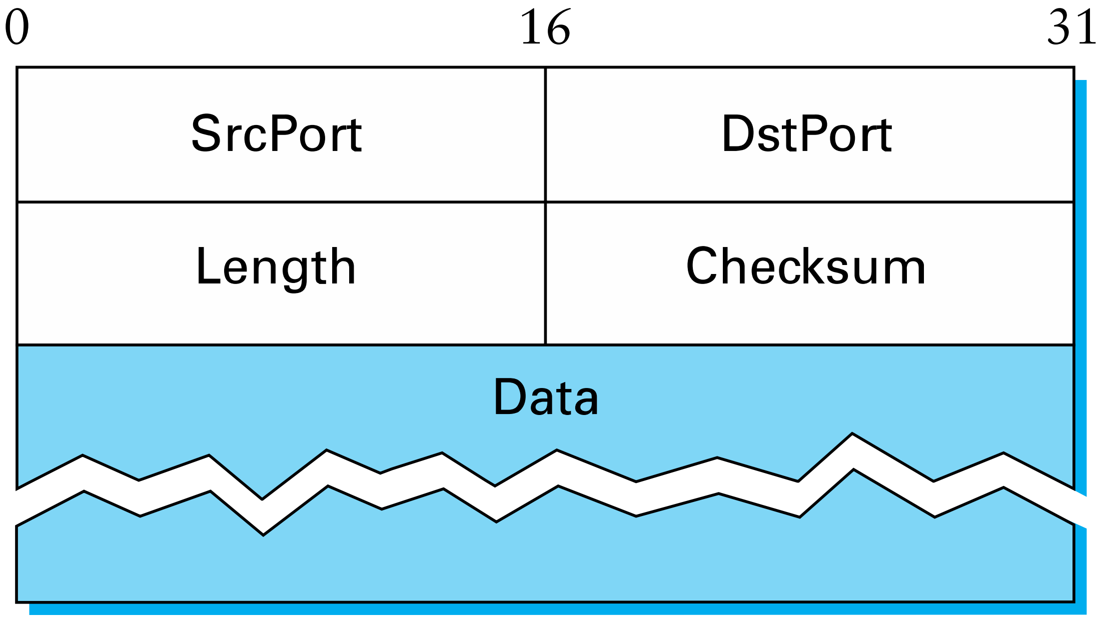
```
tkbletsc@FREEMAN ~ $ 
```

# Connectionless vs. Connection

- Connectionless transport layer
  - Very similar to plain layer 4 (IP)
  - Not much additional service provided on top
  - But less networking stack software overheads as a result
  - Standard example: **User Datagram Protocol (UDP)**
- Connection-oriented transport layer
  - Provides error-free, reliable communication
  - Like having a UNIX pipe between processes on two different machines
  - Standard example: **Transmission Control Protocol (TCP)**

# UDP – Connectionless service

- User Datagram Protocol
  - – Essentially allows applications to send IP datagrams
  - – With just slightly more encapsulation
- UDP transmits **segments**
  - – Simply 8 byte header followed by payload



0 16 31

SrcPort DstPort

Length Checksum

Data

# Ports

- Allows application-level multiplexing of network services
- Processes attach to ports to use network services
  - Port attachment is done with “BIND” operation
- Destination port
  - When a UDP packet arrives, its payload is handed to process attached to the destination port specified
- Source port
  - Mainly used when some reply is needed
  - Receiver can use the source port as the dest port in reply msg

# UDP – What it does NOT do

- NO Flow control
- NO Error control
- NO Retransmission on receipt of bad segment

- User processes must handle this
- For apps needing precise control over packet flow, error control, or timing, UDP is a great fit
  - E.g. client-server situations where client sends short request and expects short reply back; client can timeout & retry easily
  - DNS (Domain Name System): For looking up IP addr of host name
    - Client sends host name, receives IP address response

# TCP – Connection-oriented Service

- Transmission Control Protocol
  - Designed for end-to-end byte stream over unreliable network
  - Robust against failures and changing network properties
- TCP interface to user programs
  - Manages TCP streams and interfaces to the IP layer
  - Accepts user data streams from processes
  - Breaks up into pieces not larger than 64 KB
    - Often 1460 data bytes to fit in 1 Ethernet frame w/ IP + TCP headers
  - Sends each piece separately as IP datagram
  - Destination machine reconstructs original byte stream
  - Handles retransmissions & re-ordering
  - Provides error-free, reliable communication
- Result:
  - Can think of link like a pipe: Put data in one end, other side takes it out

# TCP Service Model

- TCP service setup as follows:
  - Two endpoint processes create endpoints called sockets
  - Each socket has an address: IP address of host + 16-bit **port**
  - API functions used to create & communicate on sockets
- Ports
  - Numbers below 1024 called “well-known ports”
    - Reserved for standard services, like FTP, HTTP, SMTP
      http://www.iana.org/assignments/service-names-port-numbers/service-names-port-numbers.xhtml

# TCP Service Model (2)

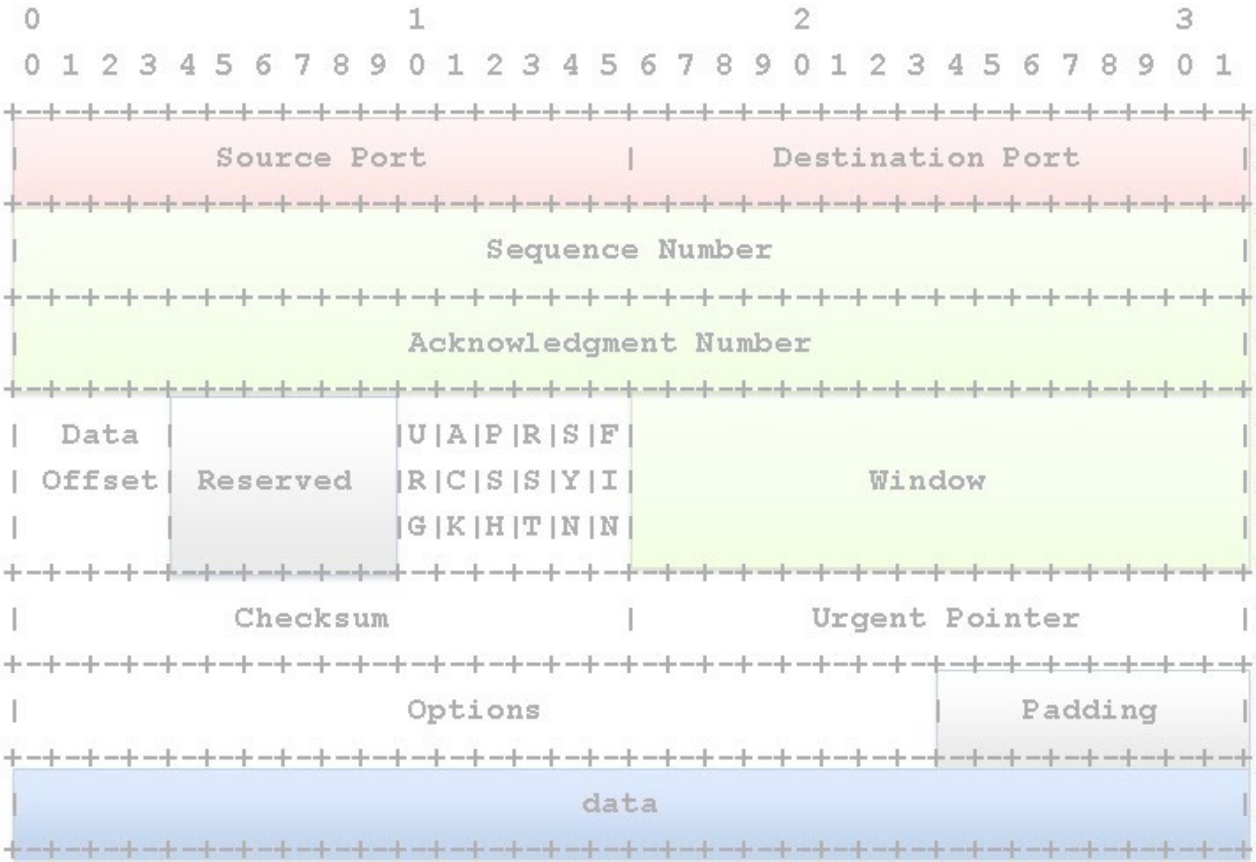
- TCP connections are full-duplex & point-to-point
  - Simultaneous traffic in both directions
  - Exactly 2 endpoints (no multicast or broadcast)
- TCP connection is a byte stream, not message stream
  - Receiver has no way to know what granularity bytes were sent
  - E.g. 4 x 512 byte writes vs. 1 x 2048 byte write
  - It can just receive some # of bytes at a time
  - Just like UNIX files!
- TCP may buffer data or send it immediately
  - PUSH flag indicates to TCP not to delay transmission
  - TCP tries to make a latency vs. bandwidth tradeoff

# TCP Protocol

- TCP sequence number underlies much of the protocol
  - Every byte sent has its own **32-bit** sequence number
- TCP exchanges data in segments
  - 20-byte fixed header (w/ optional part)
  - Followed by 0 or more data bytes
  - TCP can merge writes into one segment or split a write up
  - Segment size limitations:
    - Must fit (including header) inside 65,515 byte IP payload
    - Networks have a MTU (max transfer unit)
      - e.g. 1500 bytes for Ethernet payload size
- Uses a sliding window protocol (acks + timeout + seq #)
  - Ack indicates the next seq # the receiver expects to get
  - May be piggy-backed with data going in the other direction

We're skipping this sort of stuff. If this were a networking course,
we'd spend a looooong time on how TCP works.

# TCP Header



```
 0                   1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|          Source Port          |       Destination Port        |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                        Sequence Number                        |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                    Acknowledgment Number                      |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|  Data |           |U|A|P|R|S|F|                               |
| Offset| Reserved  |R|C|S|S|Y|I|            Window             |
|       |           |G|K|H|T|N|N|                               |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|           Checksum            |         Urgent Pointer        |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                    Options                    |    Padding    |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                             data                              |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

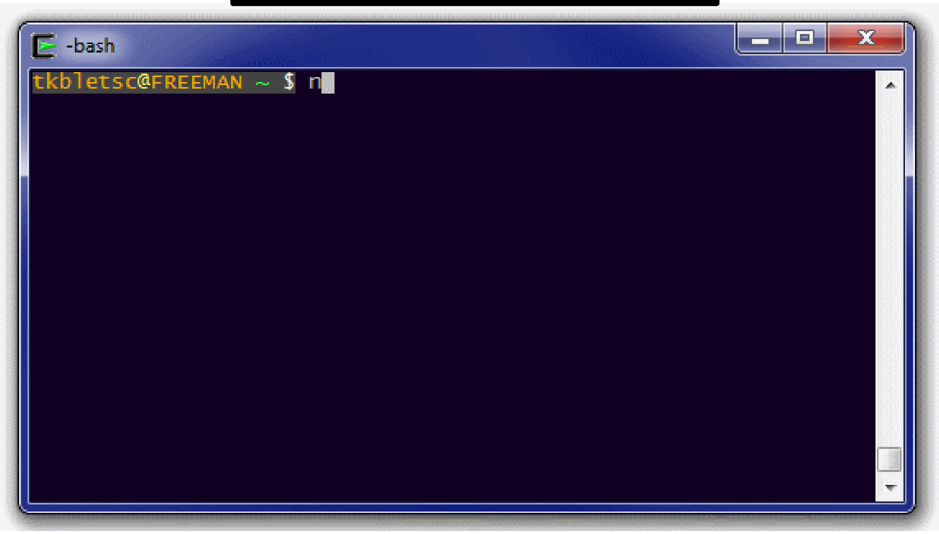
We're skipping this sort of stuff. If this were a networking course, we'd spend a looooong time on how TCP works.

# Layers 5+

Example: HTTP, lots more
(fulfill the actual use case)

- Communication of whatever you want
- Can use whatever transport(s) is(are) convenient/appropriate
- Freely structured
- Examples:
  - Skype (UDP)
  - SMTP = email (TCP)
  - HTTP = web (TCP)
  - Online games (TCP and/or UDP)

**Below:** Manually speaking HTTP to request http://google.com/ using the 'netcat' (nc) tool.

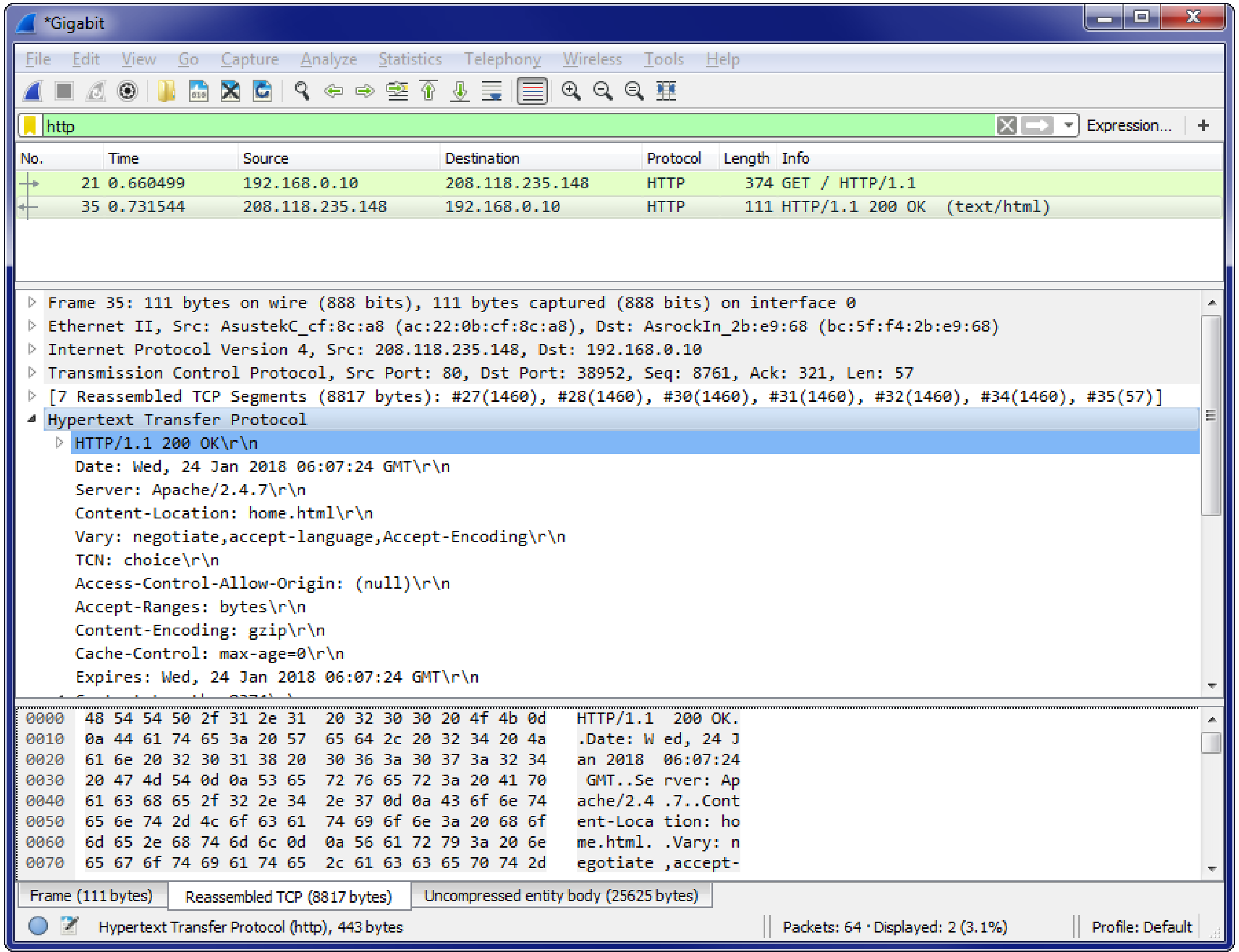


```
tkbletsc@FREEMAN ~ $ n
```

# Demo: Wireshark

- Can observe packets in transit with network sniffer, e.g. Wireshark

**Below:** Trace of a Firefox request for http://www.gnu.org/


*Gigabit
File Edit View Go Capture Analyze Statistics Telephony Wireless Tools Help
Expression...
+

Filter: http

```
No.   Time       Source           Destination      Protocol  Length  Info
   21 0.660499   192.168.0.10     208.118.235.148  HTTP         374  GET / HTTP/1.1
   35 0.731544   208.118.235.148  192.168.0.10     HTTP         111  HTTP/1.1 200 OK  (text/html)
```

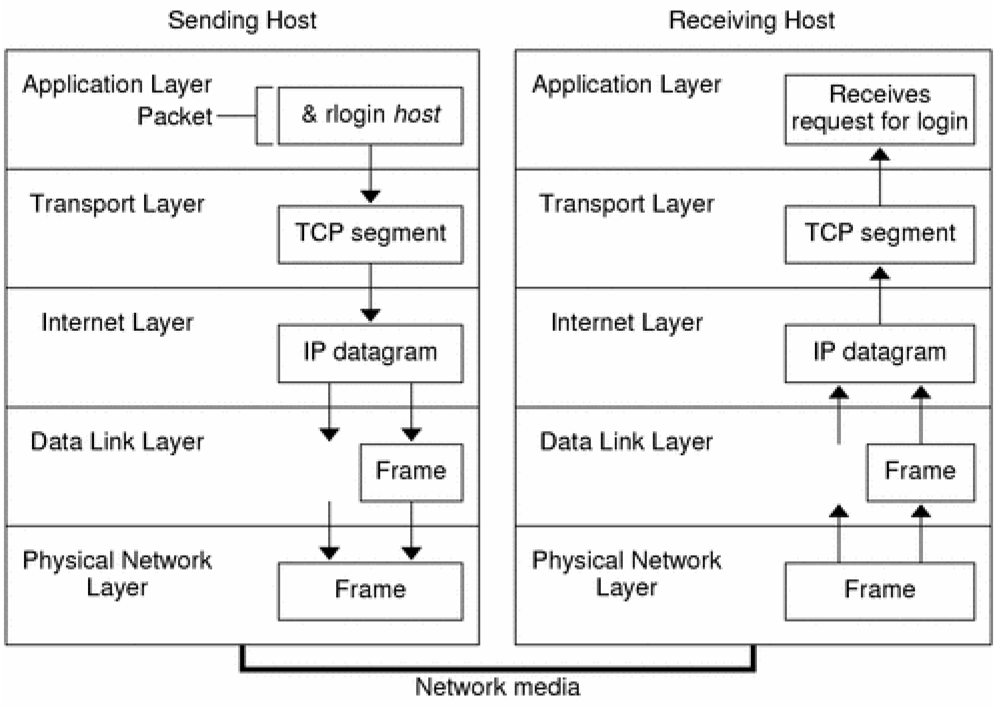
```
▷ Frame 35: 111 bytes on wire (888 bits), 111 bytes captured (888 bits) on interface 0
▷ Ethernet II, Src: AsustekC_cf:8c:a8 (ac:22:0b:cf:8c:a8), Dst: AsrockIn_2b:e9:68 (bc:5f:f4:2b:e9:68)
▷ Internet Protocol Version 4, Src: 208.118.235.148, Dst: 192.168.0.10
▷ Transmission Control Protocol, Src Port: 80, Dst Port: 38952, Seq: 8761, Ack: 321, Len: 57
▷ [7 Reassembled TCP Segments (8817 bytes): #27(1460), #28(1460), #30(1460), #31(1460), #32(1460), #34(1460), #35(57)]
◢ Hypertext Transfer Protocol
  ▷ HTTP/1.1 200 OK\r\n
    Date: Wed, 24 Jan 2018 06:07:24 GMT\r\n
    Server: Apache/2.4.7\r\n
    Content-Location: home.html\r\n
    Vary: negotiate,accept-language,Accept-Encoding\r\n
    TCN: choice\r\n
    Access-Control-Allow-Origin: (null)\r\n
    Accept-Ranges: bytes\r\n
    Content-Encoding: gzip\r\n
    Cache-Control: max-age=0\r\n
    Expires: Wed, 24 Jan 2018 06:07:24 GMT\r\n
```

```
0000  48 54 54 50 2f 31 2e 31  20 32 30 30 20 4f 4b 0d   HTTP/1.1  200 OK.
0010  0a 44 61 74 65 3a 20 57  65 64 2c 20 32 34 20 4a   .Date: W ed, 24 J
0020  61 6e 20 32 30 31 38 20  30 36 3a 30 37 3a 32 34   an 2018  06:07:24
0030  20 47 4d 54 0d 0a 53 65  72 76 65 72 3a 20 41 70    GMT..Se rver: Ap
0040  61 63 68 65 2f 32 2e 34  2e 37 0d 0a 43 6f 6e 74   ache/2.4 .7..Cont
0050  65 6e 74 2d 4c 6f 63 61  74 69 6f 6e 3a 20 68 6f   ent-Loca tion: ho
0060  6d 65 2e 68 74 6d 6c 0d  0a 56 61 72 79 3a 20 6e   me.html. .Vary: n
0070  65 67 6f 74 69 61 74 65  2c 61 63 63 65 70 74 2d   egotiate ,accept-
```

Frame (111 bytes) | Reassembled TCP (8817 bytes) | Uncompressed entity body (25625 bytes)

Hypertext Transfer Protocol (http), 443 bytes | Packets: 64 · Displayed: 2 (3.1%) | Profile: Default

# Network layer summary



Get http://pics.com/dog.jpg
(overall goal)

Transport dog.jpg data stream reliably

Send packets of data stream across world to pics.com

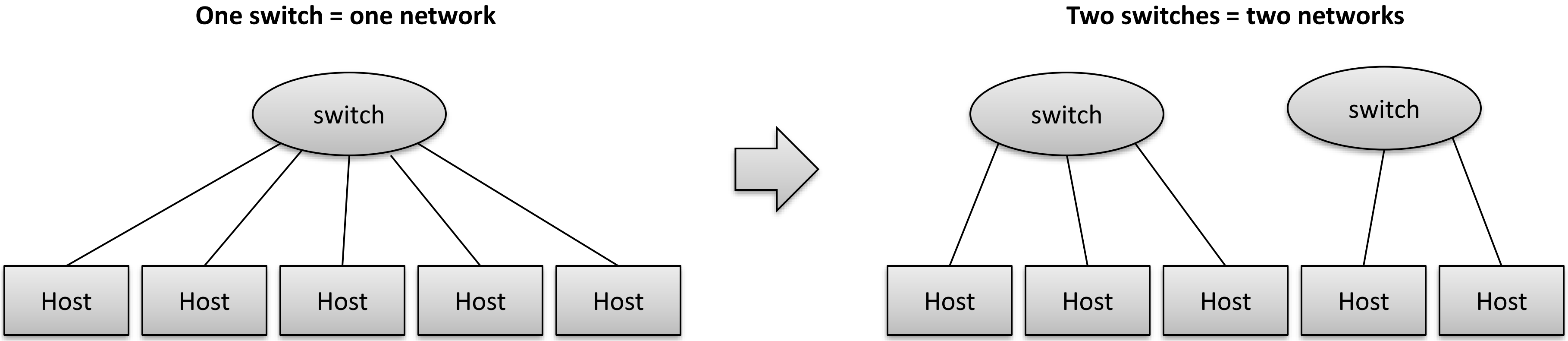
Send packet to router on my network;
I assume it can eventually reach pics.com

Put electrical pulses on wire that represent the packet

# One more thing... VLANs

- Everyone on the same **layer 2 network** is in one “broadcast domain”
  - Equals one **IP subnet** (e.g. 192.168.0.0/24 would be one network)
  - No IP routing to go point-to-point; a **network switch** delivers directly

- What if we WANT a layer 2 boundary?
  - Just buy a physical switch for each network!



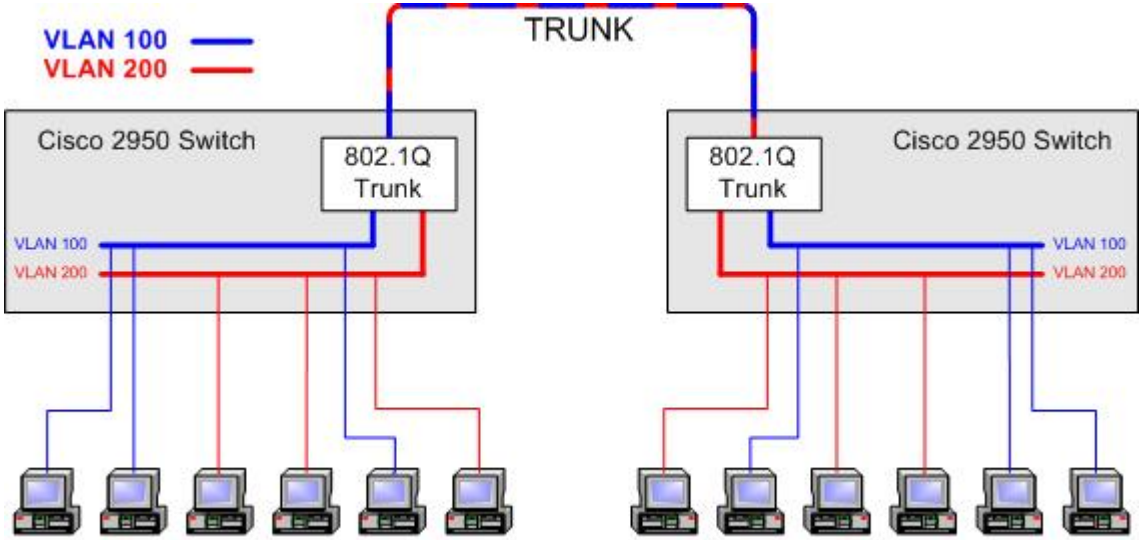
**One switch = one network**

**Two switches = two networks**

- But we don’t want to buy a different physical switch for each one!!

# One more thing... VLANs

- *Logically* separate layer 2 networks
- Switch ports can be:
  - **Access ports:** can only see one VLAN, aren't aware of VLAN concept
  - **Trunk ports:** end point includes a VLAN tag in packet header to indicate which VLAN it wants to talk to; interprets such headers on incoming packets



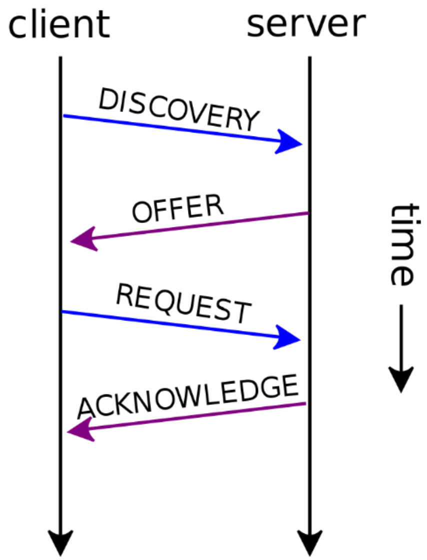
http://www.examcollection.com/certification-training/ccnp-configure-and-verify-vlans-and-trunking.html

# Dynamic Host Configuration Protocol (DHCP)

(It’s just one slide)

# Dynamic Host Configuration Protocol (DHCP)

- DHCP: Allow hosts to enter a network and ask "what IP should I use for myself?"
- How it works:
  1. Client sends an IP broadcast "DISCOVERY" request (destination 255.255.255.255 UDP port 67)
  2. DHCP server on network sends an "OFFER" with IP address and other config (gateway router, DNS servers, maybe other stuff)
     - *Note: multiple offers might be provided by multiple DHCP servers (but usually it's just one)*
  3. Client sends a broadcast REQUEST for one of the offers
  4. DHCP server sends ACKNOWLEDGE back
  5. Client now has an IP address and basic config info



- DHCP can also be used to start network-boot (PXE), commonly used for diskless clusters, OS auto-install, etc.

# Domain Name System (DNS)

(Many slides)

# Purpose of DNS

- Map an easy-to-remember name to an IP address
- Implications
  - Without DNS, to send IP packet, must remember IP addresses manually! ...and they could change!
  - With DNS, we can use the name directly:
    - www.google.com or www.cnn.com
- DNS also provides inverse look-up that maps IP address to name

# Before there was DNS...

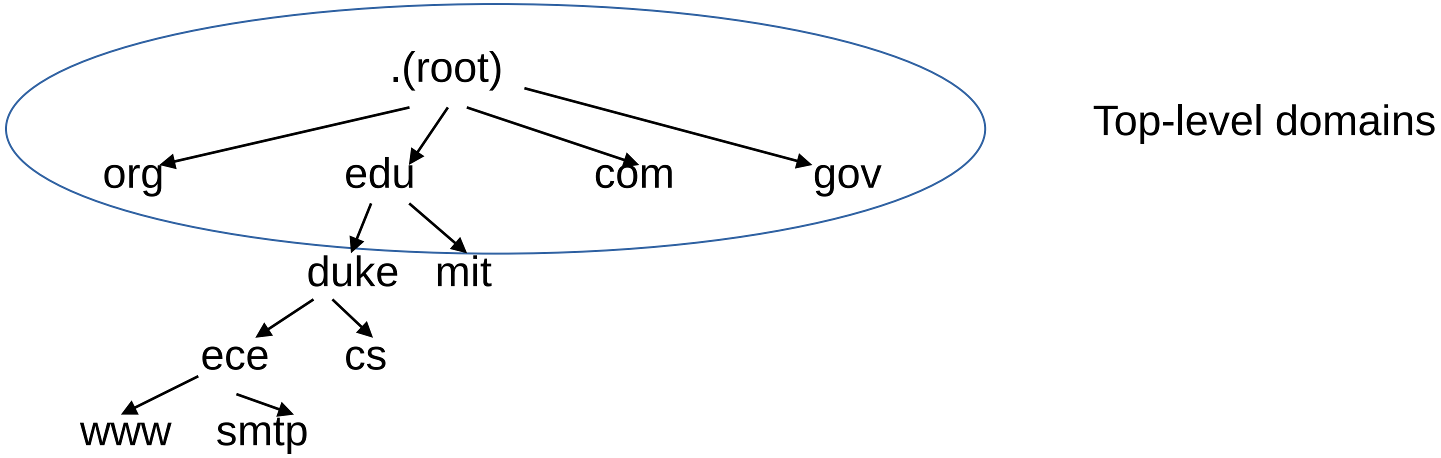
- There was the HOSTS.TXT file (on Linux today as /etc/hosts)
- Maintained at SRI Network Information Center (NIC)
- Before DNS (1985), the name-to-IP address was done by downloading this single file from a central server with FTP
  - No hierarchical structure to the file
  - Still works on most OSes; can be used to define local names

Wait, there's a file that we can change on a computer to make it think a certain DNS name points to whatever IP we say it does?

Security implications...

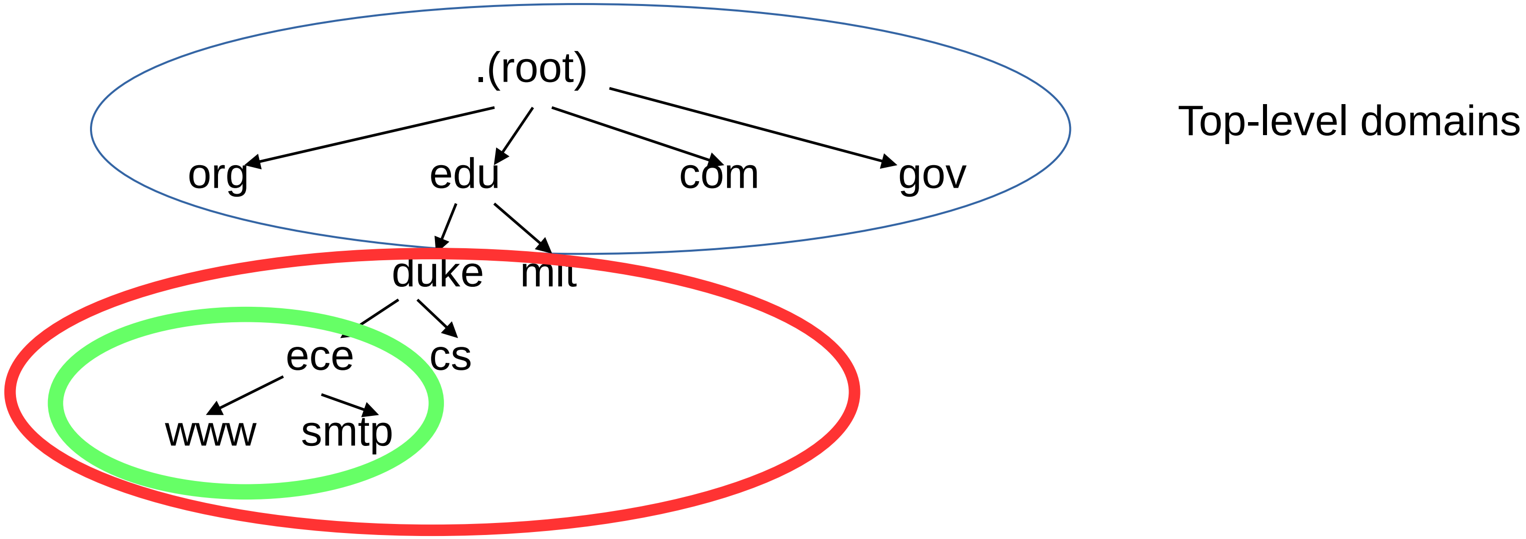
(Do the hosts + netcat demo)

# Domain Namespace



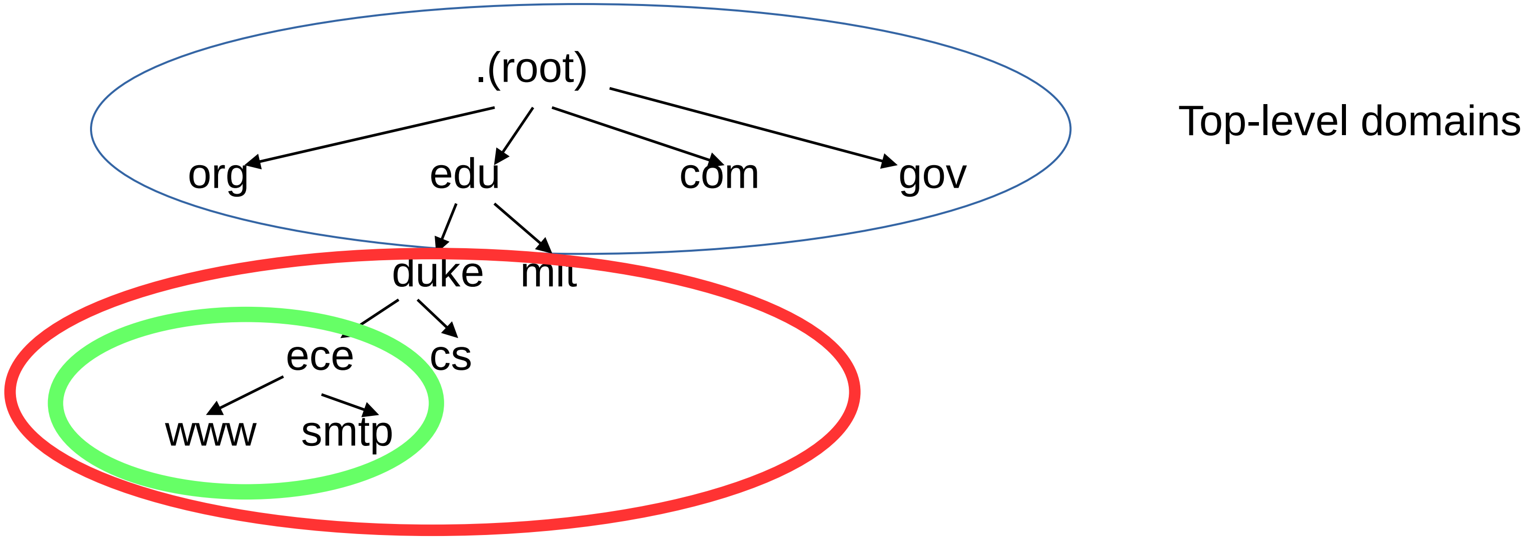
- Domain namespace is a hierarchical and logical tree structure
- Label from a node to root in the DNS tree represents a DNS name
- Each subtree below a node is a DNS domain
  - DNS domain can contain hosts or other domains (subdomains)
- Examples of DNS domains: .edu, duke.edu, ece.duke.edu

# Domain Namespace



- Red is managed by Duke
- Green is managed by ECE
- Below top-level domain, administration of name space is delegated to organizations
    - Each organization can further delegate

# Fully Qualified Domain Names



- Every node in the DNS domain tree can be identified by a FQDN
  - Fully Qualified Domain Name
- FDQN (from right to left) consists of labels (“ece”, “duke”, “edu”) separated by a period from the root to the node
- Each label can be up to 63 characters; full DNS name <= 255 chars
- FDQN contains characters, digits, dashes; not case-sensitive

# Top-Level Domains

- Three types of top-level domains:
  - Generic Top Level Domains (gTLD)
    - 3 char code indicates the function of the organization
    - Use primarily within the US (e.g. gov, mil, edu, org, com, net)
  - Country Code Top Level Domain (ccTLD)
    - 2 char country or region code (e.g. us, jp, uk)
  - Reverse Domain
    - Special domain used for IP address-to-name mapping
    - in-addr.arpa
- More than 200 top-level domains

# DNS Architecture

- Domain name space
  - Domain namespace is a hierarchical tree structure, a domain can be delegated to an organization
- Name servers
  - Domain name hierarchy exists only in the abstract
  - A host's name servers are specified in /etc/resolv.conf
- Resource records: (Name, Value, Type, Class, TTL)
  - Name (“duke.edu”) and Value (“152.3.72.104”)
  - Type specifies how the “Value” should be interpreted. Examples:
    - “NS” = Name Server: name is a domain and value is name of authoritative name server for this domain
    - “A” = Address: machine name and IP address
  - Class: record type (usually “IN” for Internet)
  - TTL: how long should the record be cached

# Resource Records

```
$TTL 86400

mylab.com. IN SOA PC4.mylab.com. admin@mylab.com. (
                       1 ; serial
                       28800 ; refresh
                       7200 ; retry
                       604800 ; expire
                       86400 ; minimum ttl
                       )
;

mylab.com.      IN      NS      PC4.mylab.com.
;

localhost               A       127.0.0.1
PC4.mylab.com.  A       10.0.1.41
PC3.mylab.com.  A       10.0.1.31
PC2.mylab.com.  A       10.0.1.21
PC1.mylab.com.  A       10.0.1.11
```

- Max age of cached data in seconds

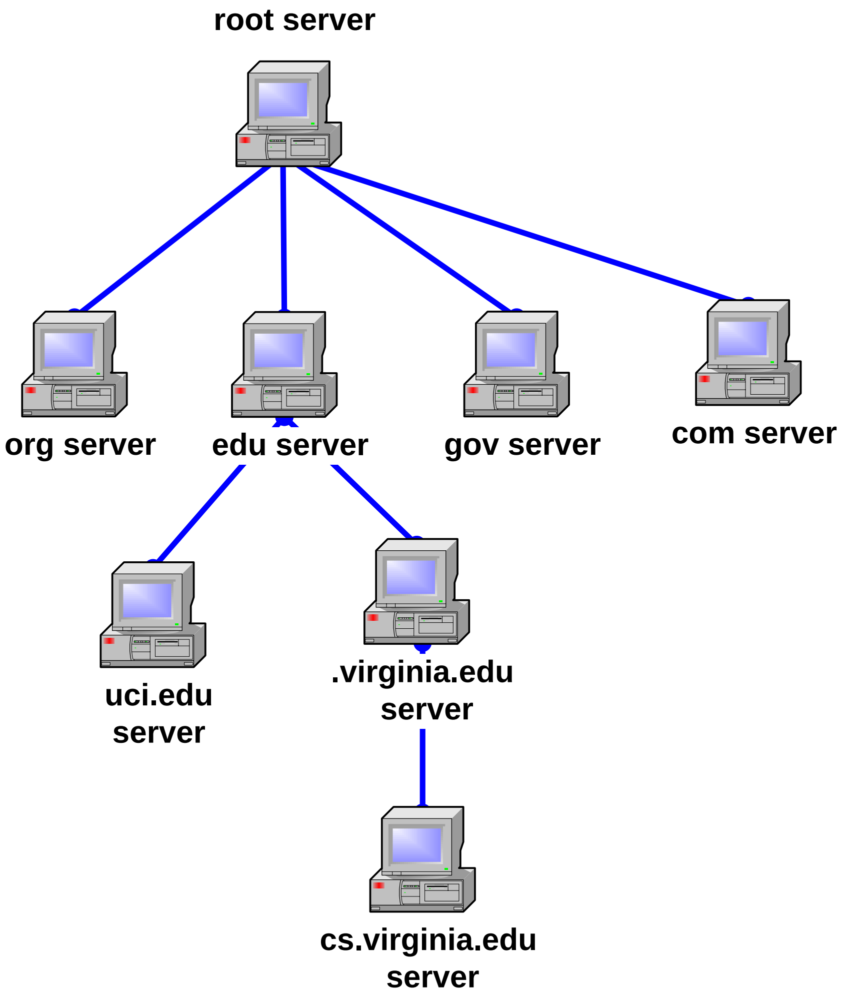
- Start of authority (SOA) record.
  Means: “This name server is authoritative for the zone Mylab.com”
- PC4.mylab.com is the name server
- admin@mylab.com is the email address of the person in charge

- Name server (NS) record: one entry for each authoritative name server

- Address (A) records: one entry for each host address

# Hierarchy of Name Servers

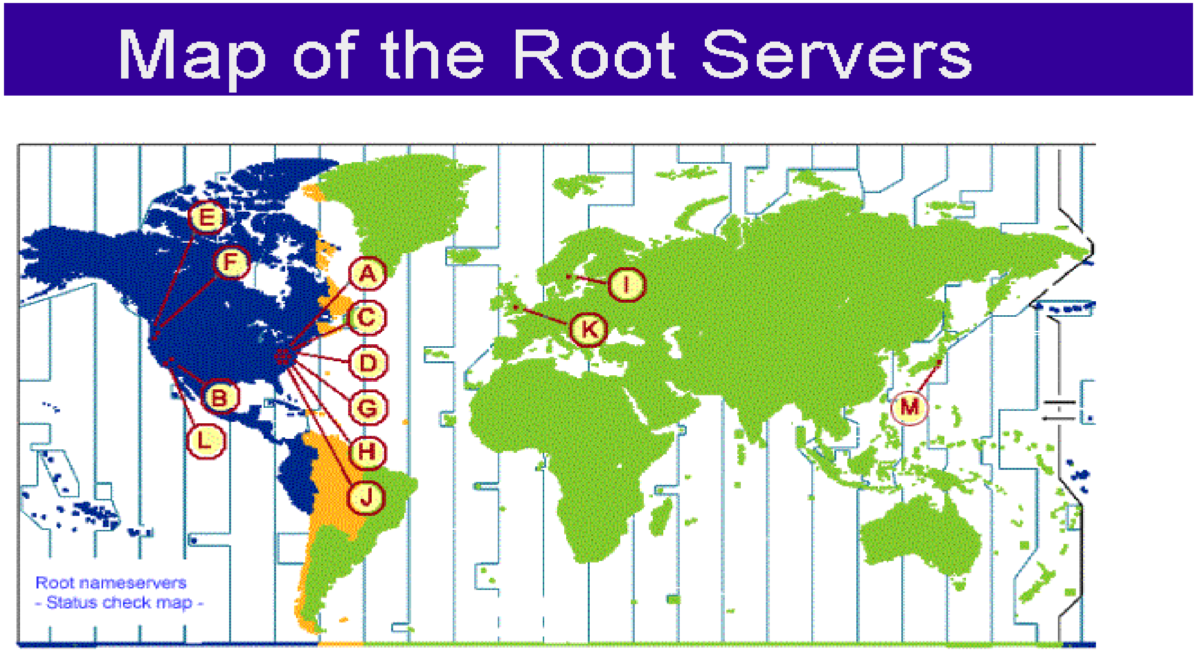
- Resolution of the hierarchical namespace is done by hierarchy of name servers
- Namespace is partitioned into zones. A zone is a contiguous portion of the DNS namespace
- Each server is responsible (authoritative for a zone)
- DNS server answers queries about host names in its zone



# Name Servers

- Each zone has a primary and secondary name server
  - For reliability
  - Primary server maintains a zone file with zone info
    - Updates made to the primary server
  - Secondary server copies data stored at the primary server
- Adding a new host:
  - When new host is added (e.g. “newmachine.ece.duke.edu”)
  - Administrator adds the IP info on the host (IP address, name) to a configuration file on the primary server

# Root Name Servers



Map of the Root Servers

- Root name servers know how to find authoritative name servers for all top-level zones
- There are 13 (virtual) root name servers
- Root servers are critical for proper functioning of name resolution

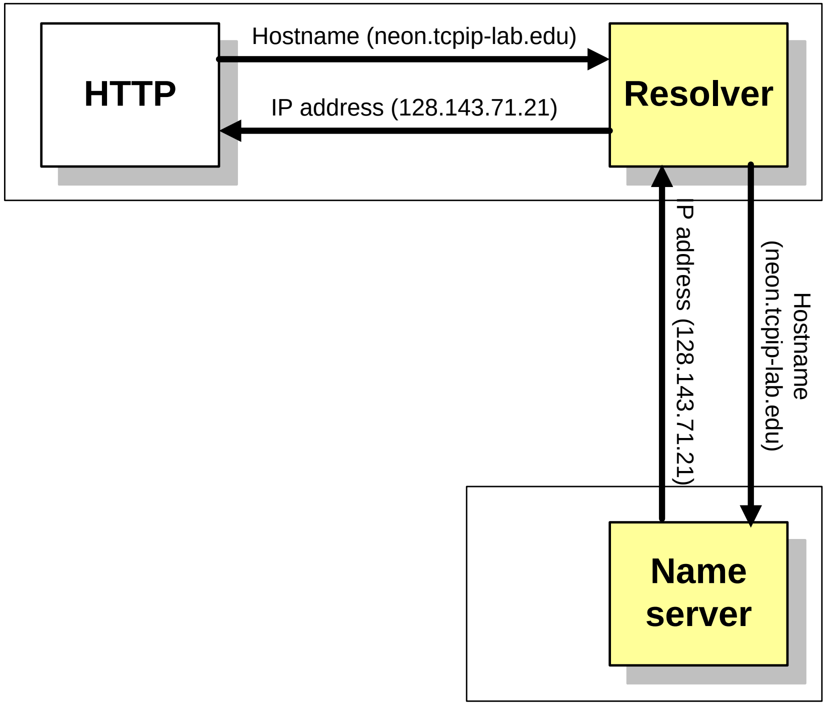
# Domain Name Resolution

1. User program issues a request for the IP address of a hostname gethostbyname()

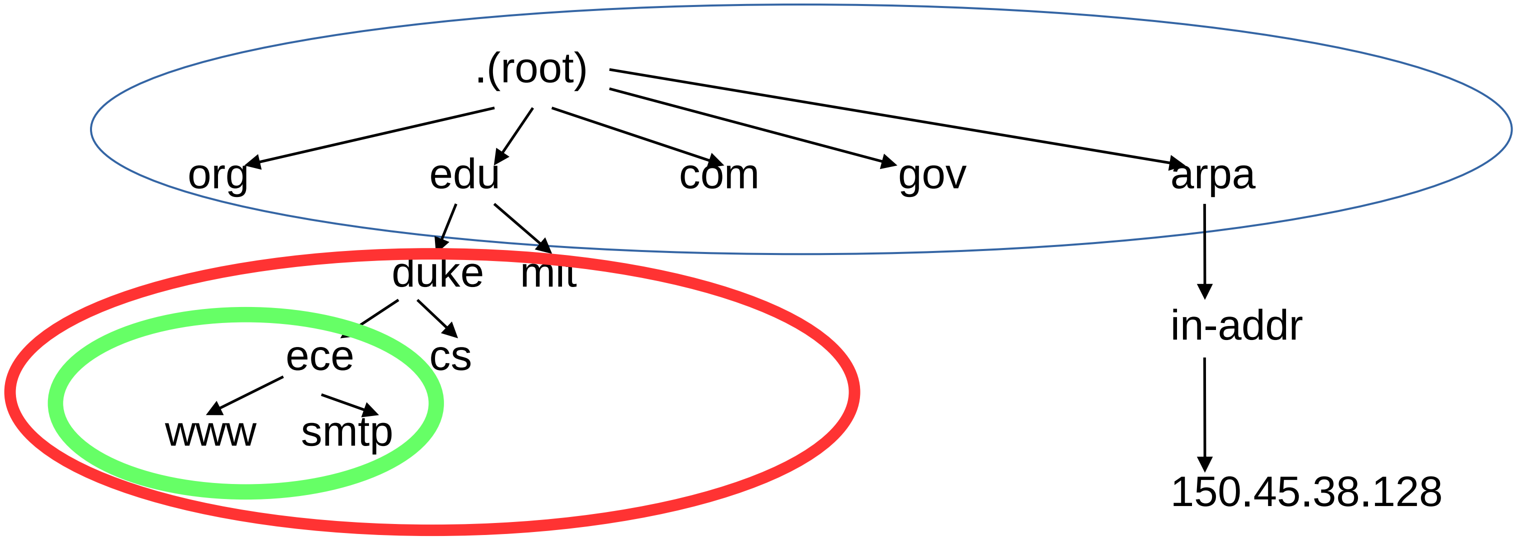
2. Local resolver formulates a DNS query to the name server of the host

3. Name server checks if it is authorized to answer the query.
   - If yes, it responds.
   - Otherwise, it will query other name servers, starting at the root tree

4. When the name server has the answer it sends it to the resolver.


HTTP
Hostname (neon.tcpip-lab.edu)
IP address (128.143.71.21)
Resolver
IP address (128.143.71.21)
Hostname (neon.tcpip-lab.edu)
Name server

# Inverse Query



- What is the host name for IP address 150.45.38.128?
  - IP address is converted to domain name: 128.38.45.150.in-addr.arpa
  - Resolver sends query for this address

# Caching

- To reduce DNS traffic, name servers cache info
  - e.g. Domain name / IP address mappings
- When entry for a query is cached, the server does not contact other servers
- Note: if an entry is sent from a cache, the reply from the server is marked as “unauthoritative”
- Caching-only servers
  - Only purpose is to cache results
  - Do not contain zone info or a zone database file
- Negative responses will be cached too:
  - Caching “name in question does not exist”
  - Caching “Name in record exists, but requested data does not”

# Modern follow-ons

- DNS with DHCP integration
  - When a new host uses DHCP to get on the network, the DHCP server can tell the DNS server about it, then the DNS server can answer requests for that host by name
- **Multicast DNS (mDNS)** and **Link-Local Multicast Name Resolution (LLMNR)**
  - Resolve hostnames when there’s no local DNS server
  - Allows “automagic” host discovery on individual networks
  - Zero configuration – they’re self-organizing protocols
- DNSSEC: DNS security (next slides)
  - Provides integrity, not confidentiality
- DNS over HTTPS (DoH): Very new standard, also provides encryption
  - Provides integrity *and* confidentiality

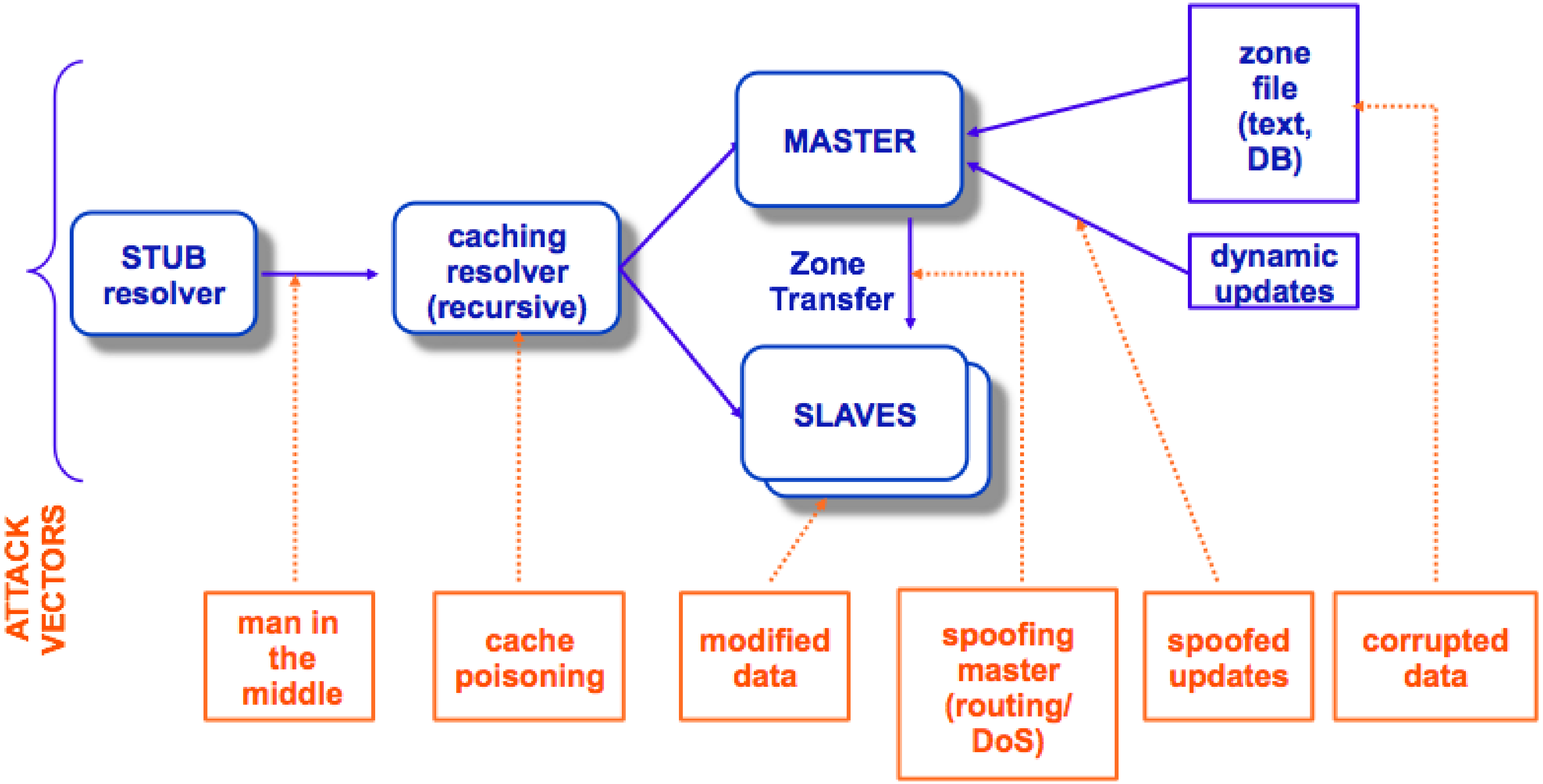
# DNSSEC

- Problem: how do I know this DNS record I got is authentic?
- **DNSSEC**: A commonly deployed protocol to provide DNS integrity
  - "Sign" the record with asymmetric cryptography,
    use a "chain of trust" to show that the signature is valid.
  - (We haven't covered these concepts yet – we'll see it when we cover crypto)

- DNSSEC *doesn't* encrypt requests
  (eavesdropping attacker can still see requests)

Adapted from "High-Level Awareness of DNSSEC" by Phil Regnauld and Joe Abley, KENIC/NSRC Workshop, 2011

# DNS security issues covered by DNSSEC



Adapted from “High-Level Awareness of DNSSEC” by Phil Regnauld and Joe Abley, KENIC/NSRC Workshop, 2011

# Quick intro to socket programming

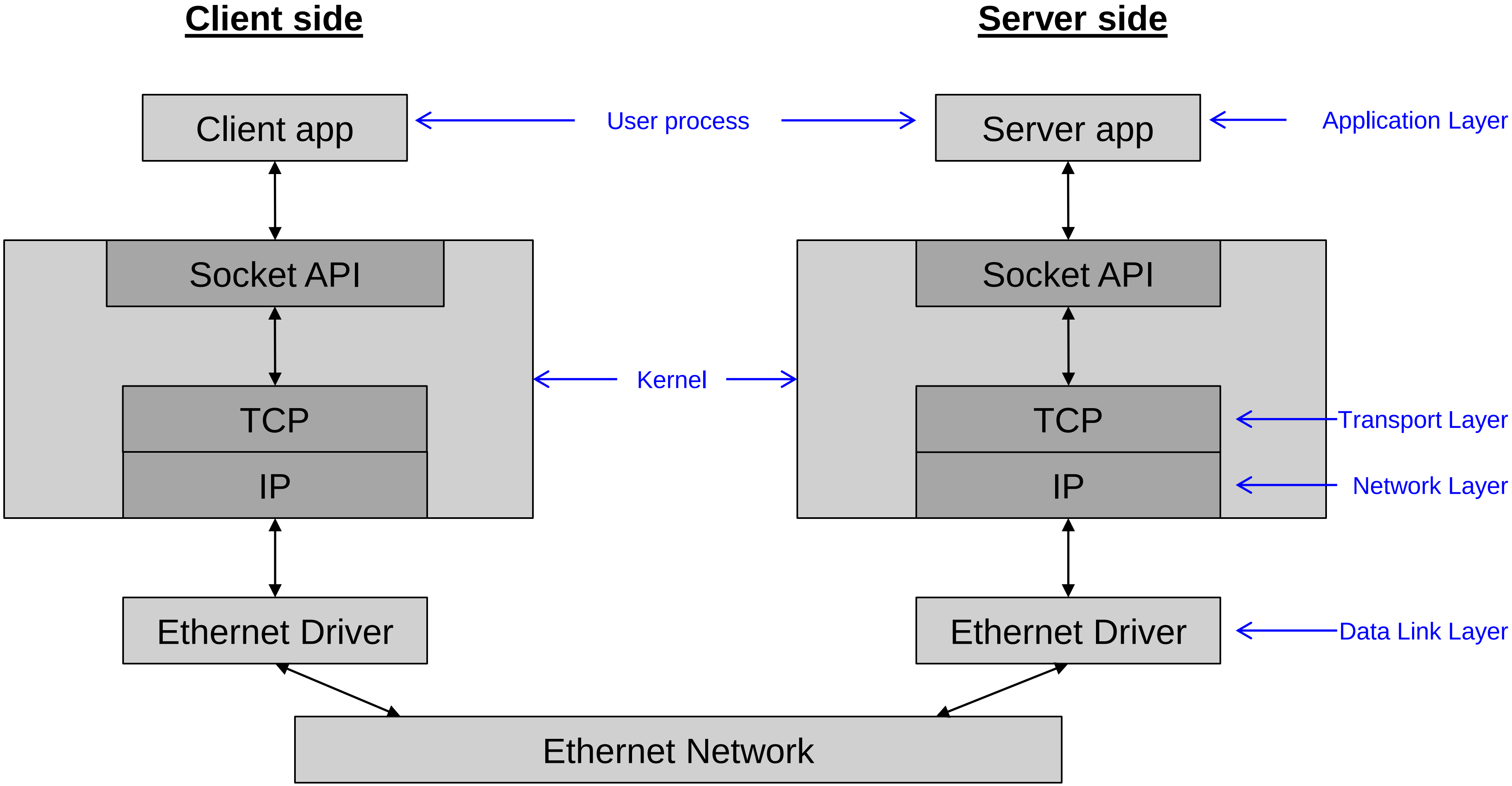
# Sockets

- How do user programs request to interact with networks?
- We can program using **network sockets**
  - For creating connections and sending / receiving messages
  - Often follows a client / server pattern

# Client-Server Model

- Common communication model in networked systems
  - Client typically communicates with a server
  - Server may connect to multiple clients at a time
- Client needs to know:
  - Existence of a server providing the desired service
  - Address (commonly IP address) of the server
- Server does not need to know either about the client

# Client-Server Overview

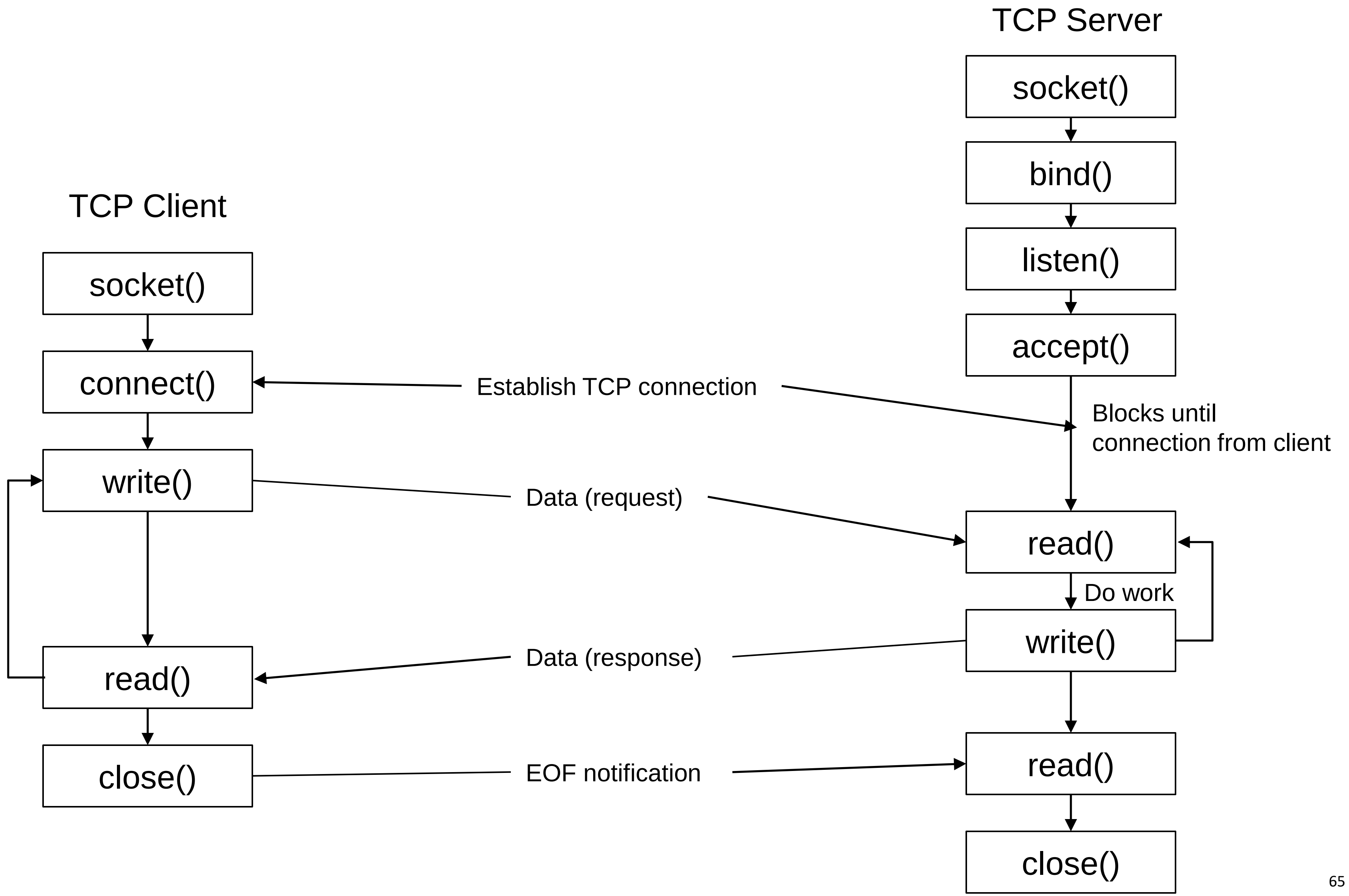


- Client and Server communicating across Ethernet using TCP/IP

# Networking concept reminders

- Network interface is identified by an IP address
  - Or a hostname, which translates into an IP address
  - E.g. 127.0.0.1, localhost or login.oit.duke.edu
- Interface has 65536 ports (0-65535)
- Processes attach to ports to use network services
  - Port attachment is done with **`bind()`** operation
- Allows application-level multiplexing of network services
  - E.g. SSH vs. Web vs. Email may all use different ports
  - Many ports are standard (e.g. 80 for web server, 22 for SSH)
  - You may have seen URLs like http://127.0.0.1:4444
    - 127.0.0.1 is the IP, 4444 is the port

# TCP Socket API



TCP Server

socket()

bind()

listen()

accept()

Blocks until connection from client

read()

Do work

write()

read()

close()

TCP Client

socket()

connect()

write()

read()

close()

Establish TCP connection

Data (request)

Data (response)

EOF notification

# Example – UNIX TCP sockets

- Let’s look at example code…
- Here is a great reference for use of socket-related calls
  - http://beej.us/guide/bgnet/

| Primitive | Meaning |
|---|---|
| `socket()` | Create a new communication end point |
| `bind()` | Attach a local address to a socket |
| `listen()` | Announce willingness to accept connections; give queue size |
| `accept()` | Block the caller until a connection attempt arrives |
| `connect()` | Actively attempt to establish a connection |
| `send()` | Send some data over the connection |
| `recv()` | Receive some data from the connection |
| `close()` | Release the connection |

# Server-Side Structure

- Often follows a common pattern to serve incoming requests

```
pid_t pid;
int listenfd, connfd;
listenfd = socket(...);

/***fill the socket address with server’s well known port***/

bind(listenfd, ...);
listen(listenfd, ...);

for ( ; ; ) {
  connfd = accept(listenfd, ...); // blocking call

  if ((pid = fork()) == 0) { // create a child process to service

    close(listenfd); // child closes listening socket

    /***process the request doing something using connfd ***/
    /* ................. */

    close(connfd);
    exit(0); // child terminates
  }
  close(connfd); // parent closes connected socket
}
```