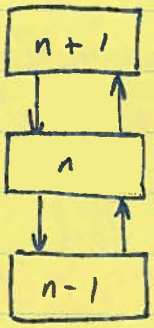
L5 Protocols, Services, Layering

5.1 Network Layers

- Network functions are complex & require careful design
- To reduce the design complexity network functions are organized as a STACK of LAYERS

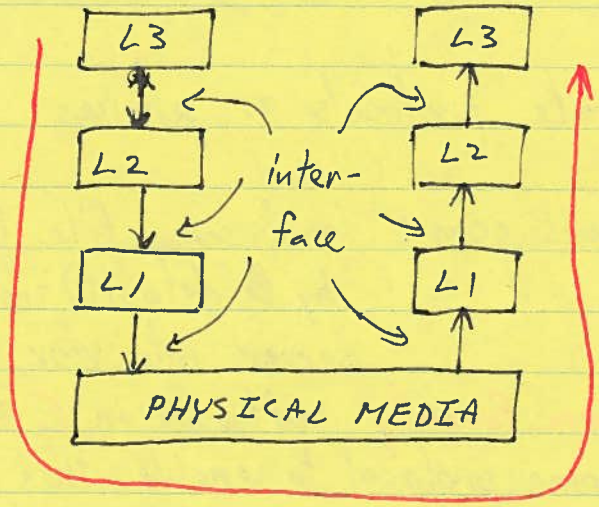


Purpose of layer is to:
1) Offer services to layer above
2) Shielding upper layers from service implementation details

(information hiding / abstract data types / data encapsulation / OOP)

5.2 Layer Communication Flow

- In comms. ea. layer passes info. to layer below until physical medium is reached (through which actual comms. takes place)

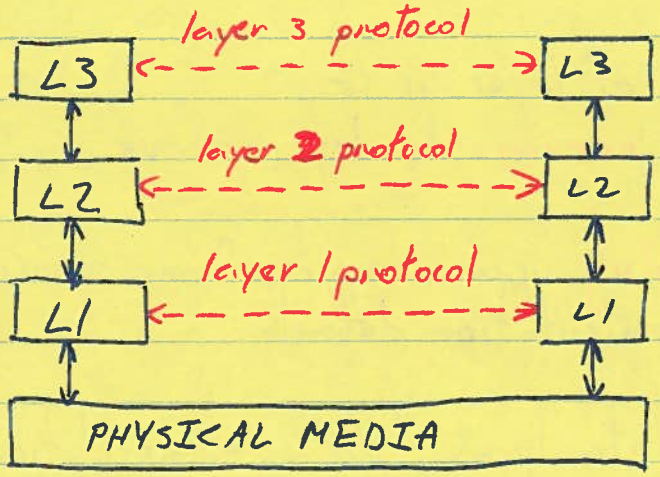


- INTERFACE lies between ea. pair of adjacent layers
- Defines which services lower layer makes available to upper one

5.3 Peers & Protocols

- Corresponding layers on different machines are called PEERS

- In the ABSTRACT we think of a virtual communication that happens between peers



- communication between peers adheres to specific rules called PROTOCOLS

- "layer n protocol"

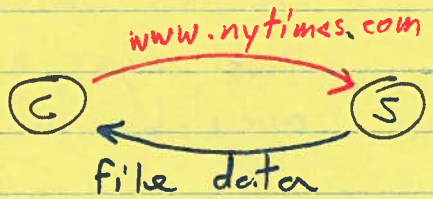
Protocols: communication rules between peer layers

protocols are implementations of a service

5.4 Web Browsing

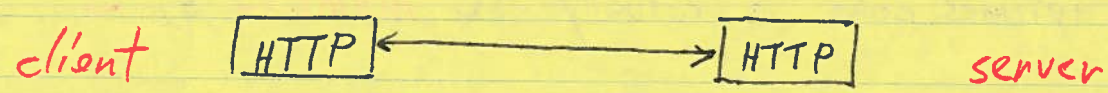
- A familiar example to illustrate protocols & services

- click on link: www.nytimes.com: load some file (index.html by default) from some server into your client



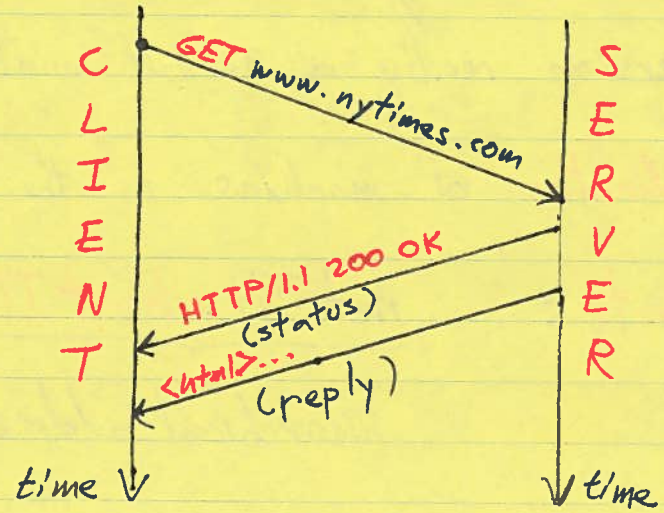
more formally: software in C & S follows some protocol to enable this data exchange

- More formally C & S use some protocol to exchange data
- HTTP (Hypertext Transfer Protocol)
- More abstract... you are engaging in a virtual comm. between two HTTP entities (using HTTP protocol)



5.5 HTTP

- To get web page specific set of (HTTP) rules is followed (a REQUEST-REPLY protocol)



the rule/protocol: an understood series of exchanges

- BUT for this to work... services of a bunch of other protocols layers/entities must be invoked
- let's look at some of these details

5.6 DNS

- domain name system
- how does C know WHERE ??? to send its request
- www.nytimes.com is actually the ADDRESS of S
- this alphanumeric version is easy for us to remember
- but machines use numerical addresses .. IP address

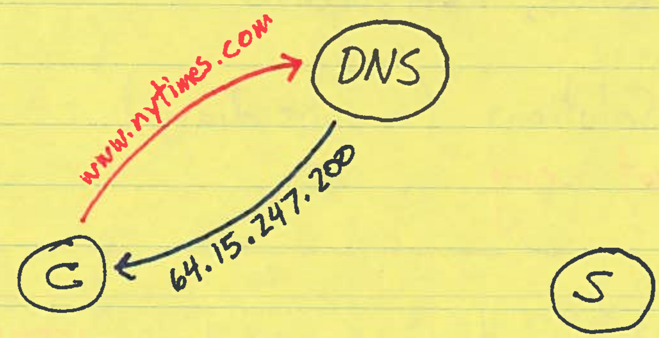
  www.nytimes.com = 64.15.247.200

  dot decimal version, really a 32 bit number
- gives logical coordinates of machine in the network
- split into 2 parts: net ID . host ID

  hierarchical addressing
- in ~~and~~ order to find IP address .. your computer must access a distributed database

  that maps alphanumeric address to numerical IP address

~~so magic~~ so, still somewhat magically... ~~we~~ we achieve



in more detail -

1) At setup DHCP (in your network router) senses your ETHERNET frames & notices you (i.e. your PHYS. ADDRESS) does not have an IP address associated with it

2) DHCP gives you IP addr. and tells you address of NAME SERVER local DNS server (name server)

3) local DNS server either has the IP mapping you are looking for -OR- knows address of distributed DNS database that will allow you to ITERATIVELY re-form the IP addr. you are looking for (~~ma~~ name server does iterative finding work)

notes: DHCP is Application Layer protocol running on UDP

DNS : 13 root servers >300 actual machines (386 in Jan. 2014)
- ICANN (Intl. Corp. for Assigned Names & Numbers) oversees
- but ultimate control at U.S. Dept. of Commerce?

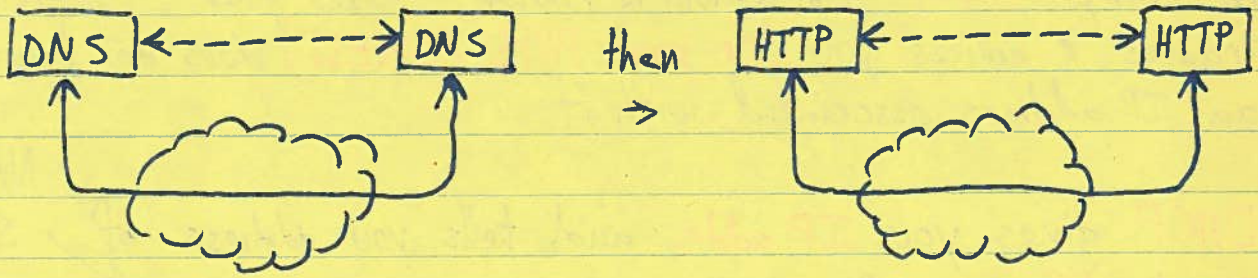
lets you find ~~domains~~ TOP level Domains { .com, .org, .net ... countries ~200? + 7

- top level domains managed by mix of governments, companies, non-profits

e.g. ~~Verisign~~ Network Solutions (a subsidiary) manages .com .net .org

---

- DNS is another protocol... at same level as HTTP (appl. layer on UDP)



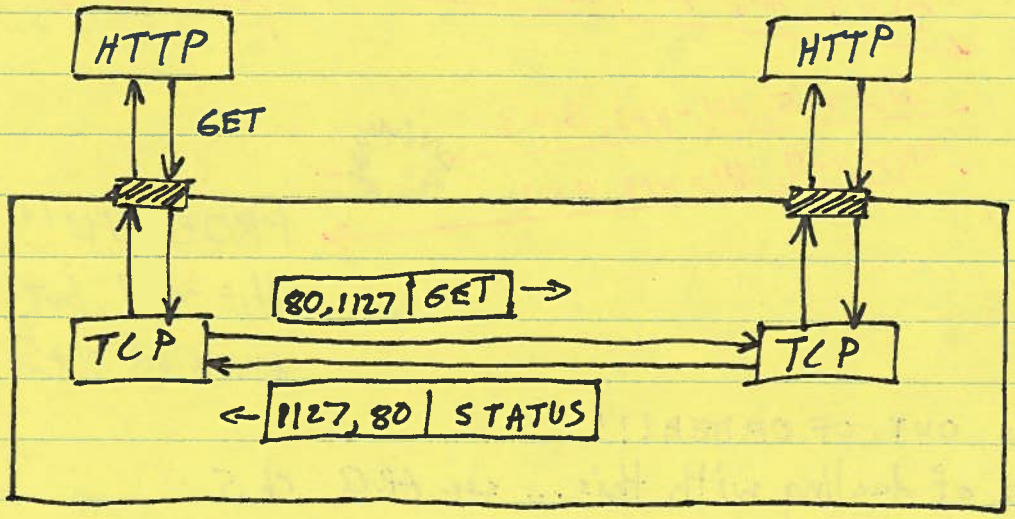
5.7 TCP Ports

- To get its message across HTTP employs the services of a lower-level protocol: TCP

- TCP can be conducting a number of protocols simultaneously
- Each protocol ~~at~~ can be running on a different process

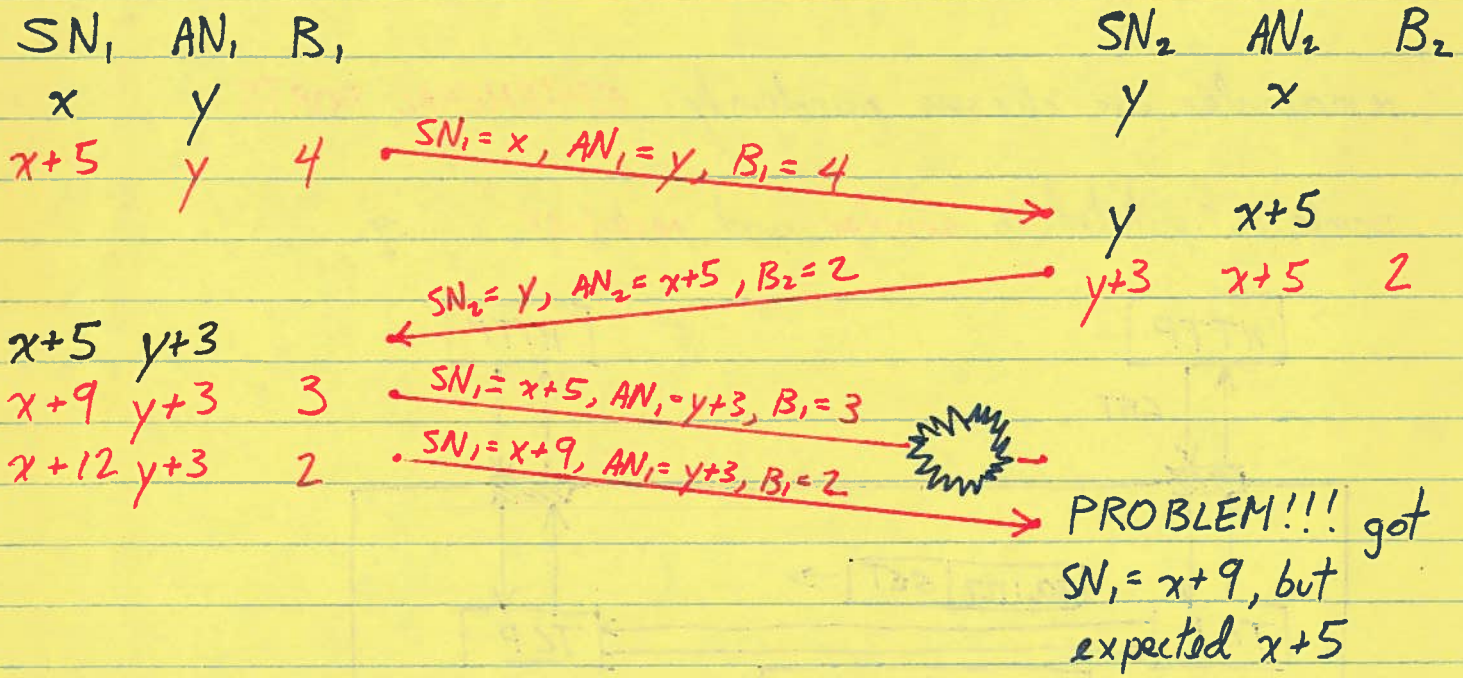
- We need some way of letting TCP know different protocol locations
- Define TCP (TRANSPORT) ADDRESS to which process can listen for incoming messages

- these addresses are called PORTS
- TCP uses 16-bit number to denote a port 64k possible
- First 1k (2^{10}) are reserved
- remainder are chosen randomly: EPHEMERAL PORTS
- ports stipulated in sender and receiver e.g.



5.8 TCP Communications

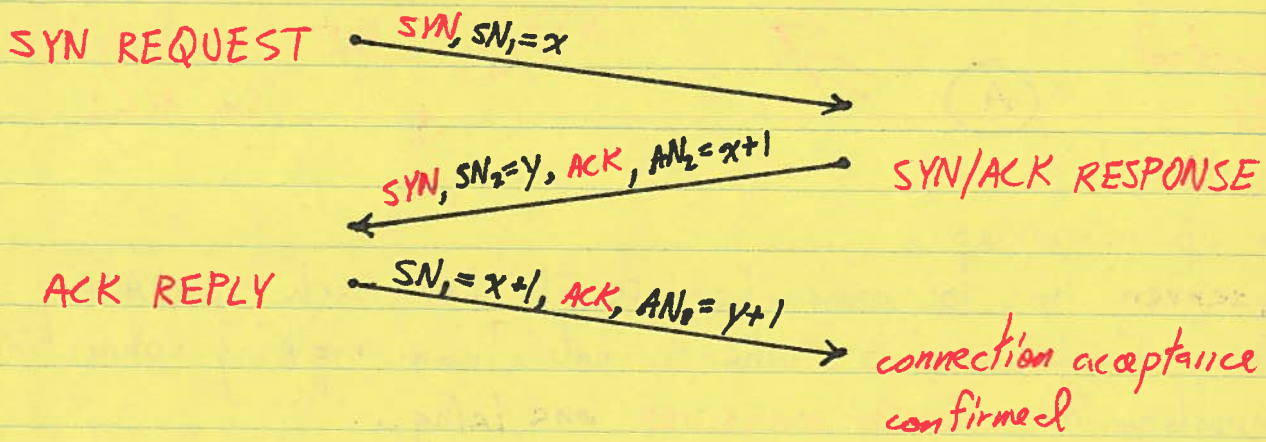
- If assigning ports is all that TCP did it would still be pretty useful (NAT... network address translation)
- But it does much more
- MOST IMPORTANTLY it provides a connection-oriented byte stream service
  - 1) Makes sure a ~~connect~~ destination is available and reserves our access to it until disconnect
  - 2) Delivers byte constituents of our service in order
- how? labels each message with a sequence number
- seq. & ack. #'s somehow initialized (next section)



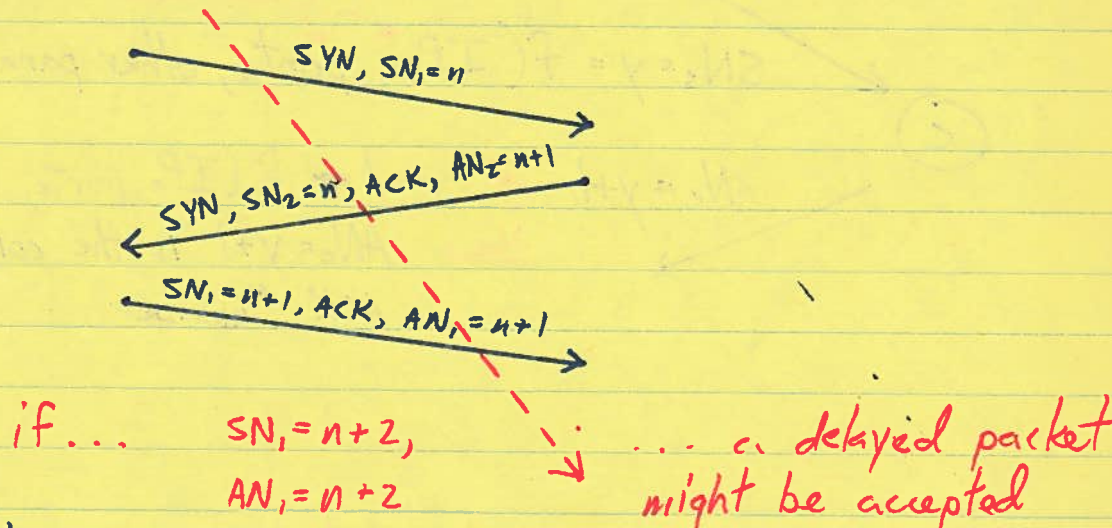
- something is OUT OF ORDER!!!
- many ways of dealing with this... see ARQ Ch.5

5.9 TCP Communications Establishment

- special measures are taken to establish a connection
- A 3-way handshake



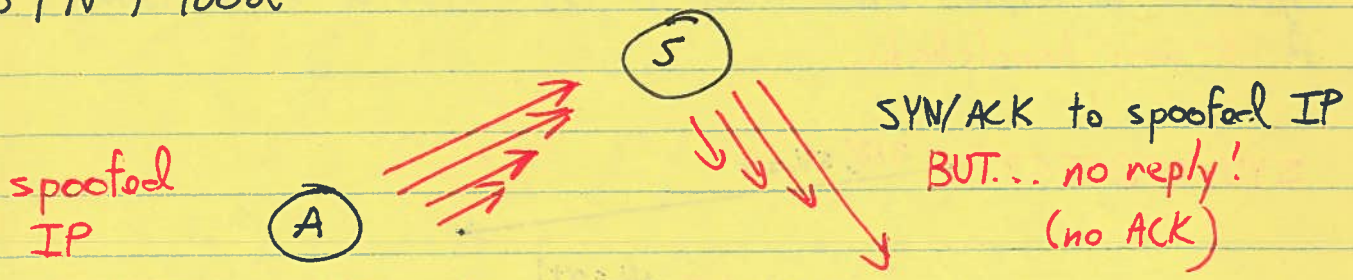
- SYN & ACK bits set to indicate that this is part of a connection establishment process
- if your initial sequence numbers are SUFFICIENTLY UNIQUE you safely setup a connection
- if numbers are somehow predictable or same e.g. ...



- similar delay problems occur even more easily with less robust handshaking schemes

5.10 TCP Hacks

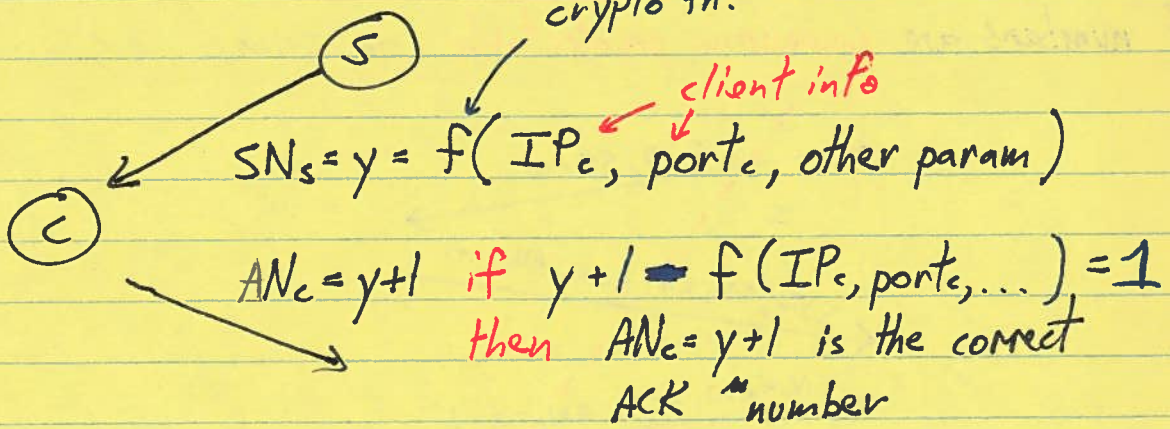
SYN Flood



- eats up resources
- S (server) has to remember its SN for each SYN/ACK
- memory load prevents other clients from making connection
- a number of countermeasures one being...

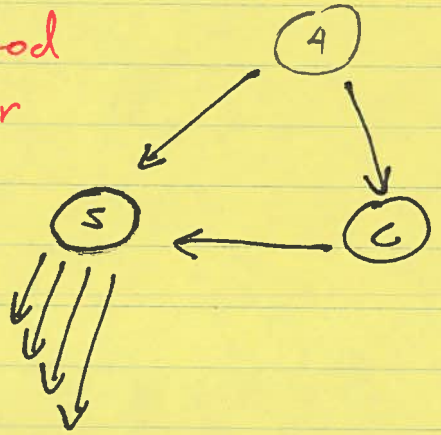
... SYN Cookies

- Instead of memorizing $SN_s = y$ in SYN/ACK...
- Just CALCULATE it with a cryptographic fn....
  ... and forget about it (... so no memory load)



Man-in-th-Middle (Mitnick Attack)

1) flood server



2) SYN to client with serves spoofed IP

3) client's SYN/ACK not acked because trusted server flooded

4) A ack's C's SYN/ACK (because you figured out what SN_C it used in it's SYN/ACK)

5) Now C trusts A's signal... echo "++" >> ~/.rhosts

... write ++ into .rhosts (without deleting anything else in .rhosts as achived with >> rather than >)

tells system to trust anybody... anyone can login & remotely execute commands

6) A send RST to S to unflood it

in general

$A \rightarrow B$: $SYN(ISN_A)$, $SRC=C$: call B pretending to be C
$B \rightarrow C$: $SYN(ISN_B)$, $ACK(ISN_A)$: B responds to C (which won't reply)
$A \rightarrow B$: $ACK(ISN_B)$, $SRC=C$: ack B (still pretending to be C) because you know B's ISN
$A \rightarrow B$: $ACK(ISN_B)$, $SRC=C$, nasty-data

(now send nasty data to B since it opened a communication with you