CSCI-1680
Wrap-up Lecture

Rodrigo Fonseca

With some material from Jen Rexford

Administrivia

- **Today is the last class!**
- **Two more things to go:**
  - Final project, May 10th
  - Final Exam, May 13th, 9AM
- **How do you study?**
  - Any covered topic is fair game, but more emphasis on content given *after* midterm (TCP on)
  - Lecture slides, homeworks, plus relevant sections of the book (will update website)
  - If in doubt, no topic not covered in class will be on the exam

What you (hopefully) learned from this course

- **Skill: Network programming**
  - C programming (most of you)
  - Socket programming
  - Server programming
  - Implementing protocols
- **Knowledge: How the Internet Works**
  - IP Protocol suite
  - Internet Architecture
  - Applications (Web, DNS, P2P, …)
- **Insight: key concepts**
  - Protocols
  - Layering
  - Naming

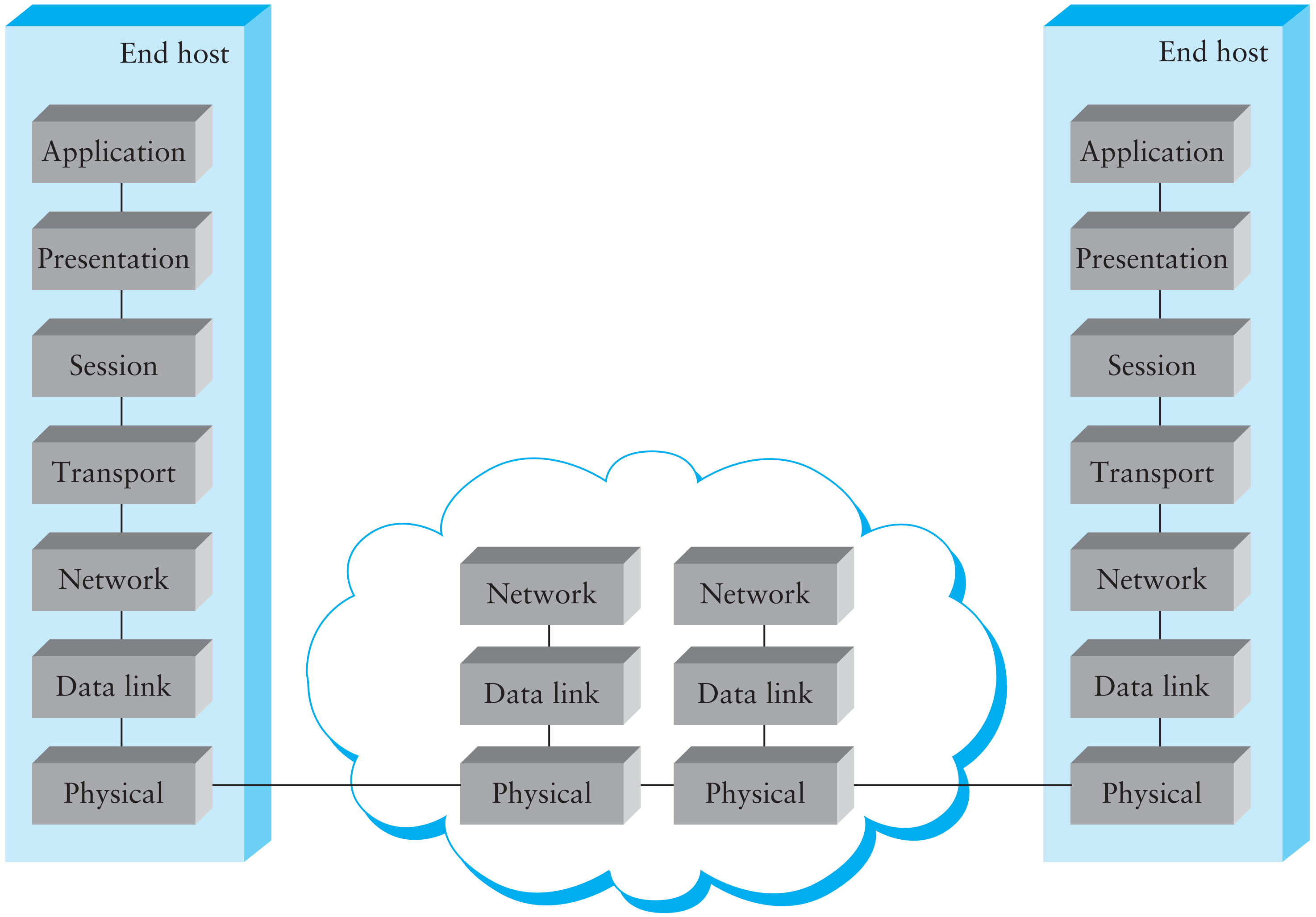
Today

- **Cut across protocols, identify *principles***
- **Internet Architecture**
  - Virtues and challenges going forward!

Networking Principles

- **We saw many layers and protocols, but some principles are comon to many**
- **Some are general CS concepts**
  - Hierarchy
  - Indirection
  - Caching
  - Randomization
- **Some are somewhat networing-specific**
  - Layering
  - Multiplexing
  - End-to-end argument
  - Soft-state

Layering

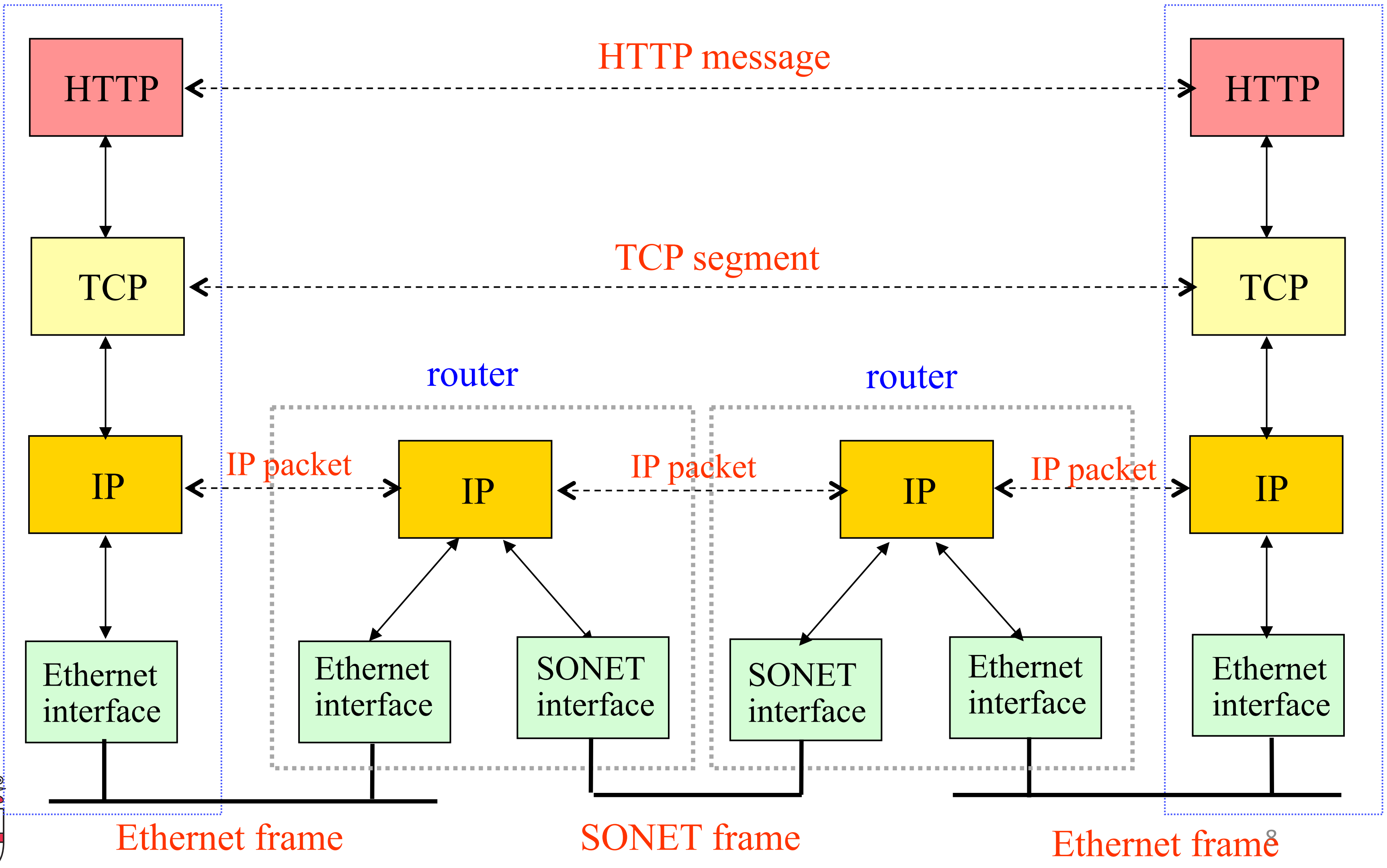


One or more nodes
within the network

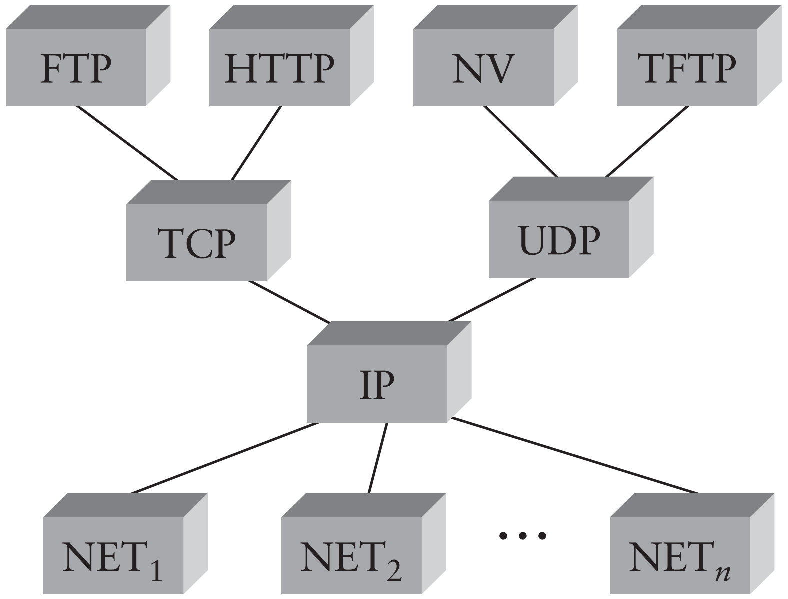
Layering

- **Strong form of encapsulation, abstraction**
- **Each layer has *three* interfaces:**
  - Services provided to upper layer
  - Protocol to communicate with peer at the same layer
  - Using the services of the lower layer
- **Provided interface hides all details of internal interface and lower layers**
- **Can be highly recursive**
  - E.g., IP over DNS, File system over Gmail!

Layering on the Internet



Layering: IP as a Narrow Waist



- **Many applications protocols on top of UDP & TCP**
- **IP works over many types of networks**
- **This is the "Hourglass" architecture of the Internet.**
  – If every network supports IP, applications run over many different networks (*e.g.*, cellular network)

Layering: Data Encapsulation

- **One layer's data is the (opaque) payload of the next**

  Stream (Application)

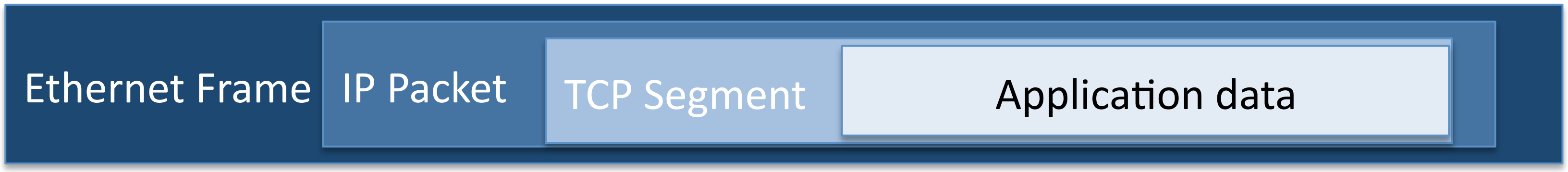
  Segments (TCP)

  Packets (IP)

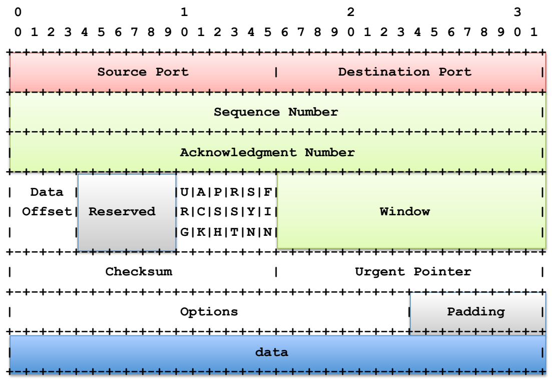
  Frames (Ethernet)

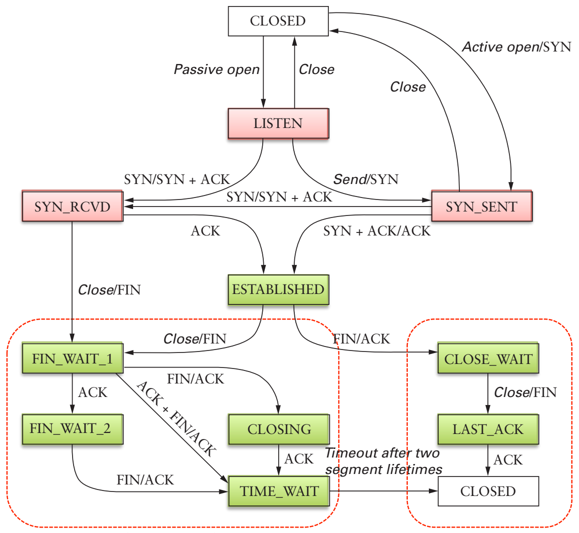
  Encoding: bits -> chips

  Modulation: chips -> signal variations



Protocols





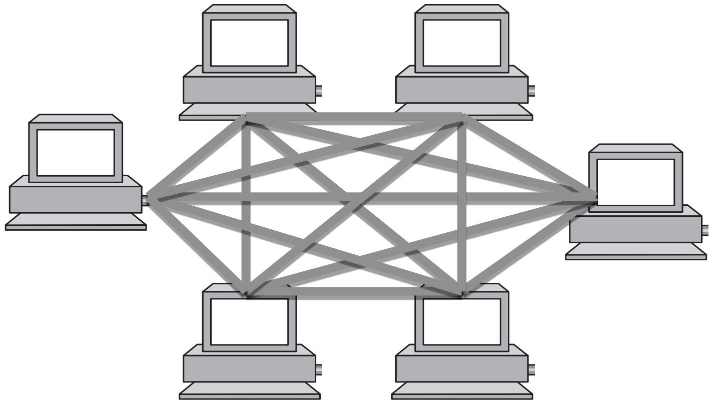
- **Specifications for communication**
  - Data formats
  - Behaviors (FSMs)
- **Allow**
  - Interoperability
  - Independent implementations
  - Don't need to specify everything
    - E.g., TCP Congestion Control
- **Postel's Robustness Principle**
  - "Be liberal in what you accept, and conservative in what you send" (RFC 1122)

Multiplexing

- **Multiple streams/flows can share at different levels**
  - Important to be able to de-multiplex: need naming
- **Sharing**
  - Cost: infrastructure sharing
  - Access: single channel sharing
  - Reuse: Implementation sharing

Multiplexing: Cost

- **Multiple flows/streams can share a link/path**
  - Packet switching
  - Circuit switching
- **Issues**
  - Coordinate access
    - In time, in space, in frequency
  - De-multiplexing (name, id)



Multiplexing: Access

- **Sharing a single channel**
- **E.g.,**
  - NAT: multiple nodes share a single IP address
    - De-multiplexing: NAT uses 5-tuple to disambiguate
  - SSH port forwarding
    - Only port 22 is open, can tunnel other ports
    - ssh other.host.com –L 5900:other.host.com:5900
  - VPN

Multiplexing: Reuse

- **No need to re-implement functionality**
  - Several streams/flows can use the services of a protocol
- **E.g.:**
  - IP/ARP/AppleTalk on Ethernet: demux EtherType
  - TCP/UDP/DCCP/… on IP: demux Protocol ID
  - HTTP/SIP/SMTP/… on TCP/UDP: demux on Port
  - Multiple hosts on one HTTP server: demux on Host: field

End-to-End Argument

“The end knows best”

“The function in question can completely and correctly be implemented only with the knowledge and help of the application standing at the end points of the communication system. Therefore, providing that questioned function as a feature of the communication system itself is not possible. (Sometimes an incomplete version of the function provided by the communication system may be useful as a *performance enhancement*.)”

End-to-end arguments in system design. Saltzer, Reed, and Clark. Technology (100), 1984

End-to-end argument

- **Reliability:**
  - File transfer application: even through TCP, has to check for file correctness (e.g., BitTorrent hashes)
  - So reliability by the transport layer is redundant
  - Would work even on top of UDP
- **Multihop wireless:**
  - 10 hops, 10% chance of packet loss per hop:
  - Chance of success: $0.9^{10} \sim 35\%$
  - If you do up to 3 retransmissions per hop, loss drops to $0.1^3 = 0.001$ per hop => chance of success $0.999^{10} \sim 99\%$!
    - But can't make it 0 => still require end-to-end check!

End-to-end argument: examples

- **Encryption: need assurance of no tampering, can only prove if the end point encrypted the message**
  - Doesn't matter if the network automatically encrypts
- **Duplicate suppression**
  - "Don't click submit twice": even though TCP is underneath, two requests will be different data for the network, application must enforce at-most-once
- **Automatic recovery**
  - Airline reservation: rather than guaranteeing that every request can survive system crashes, rely on operators to retry.

End-to-end argument

- **Instinctively we like modularity and clean interfaces**
  - Which means putting functionality in low-level abstractions
- **Examples: reliability, in-order delivery, security**
- **But some applications won't be able to rely on this**
  - Low level functionality might be redundant
  - Or might be insufficient
  - Or might be useless for some applications
  - Or might be harmful – e.g., real-time audio over a reliable, in-order delivery channel
- **Use as a guiding principle or where to place functionality**

Hierarchy

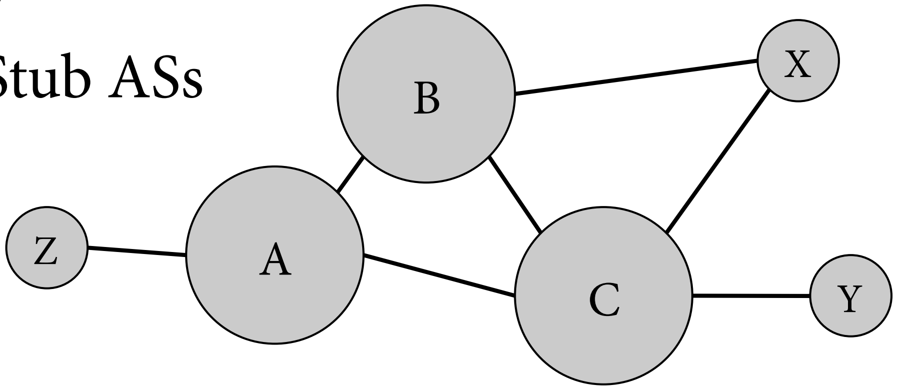
- **Scalability of large systems**
  - Cannot store all information everywhere
  - Cannot centrally control every component
- **Hierarchy as a way to manage scale**
  - Divide large system in smaller pieces
  - Summarize information about each piece
- **Hierarchy as a way to divide control**
  - Decentralized management of pieces
- **Many examples of hierarchy in the Internet**

Hierarchy Examples: IP Routing

- **IP Addressing**
  - Hierarchical assignment of address blocks
  - IANA -> Regional Internet Registries -> ISPs
  - Decentralized *control*
- **Topology**
  - (Roughly) correlated with addressing
  - Allows aggregation (CIDR)
    - Brown owns 128.148.0.0/16
  - Decreases size of routing tables!

Hierarchy Examples: IP Routing

- **AS-level Topology**
  - Separates intra and inter-domain routing
  - ASs have own economic interests
  - Delegation of control
    - Policy in inter-domain routing
    - Complete control of intra-domain routing
- **Hierarchical Topology**
  - Transit, Multi-homed, Stub ASs



Hierarchy Examples: DNS

- **Hierarchical name database**
- **Allows delegation of control**
  - Each organization controls a sub-tree
  - May delegate control
- **Allows scaling of the infrastructure**
  - A DNS server only needs to know about its sub-domains

Hierarchy Example: MAC Addresses

- **Ethernet MAC addresses are globally unique identifiers**
  - First 3 bytes: manufacturer, allocated by consortium
  - Last 3 bytes: allocated by manufacturer

Indirection

- **Referencing by name**
- **"Any problem in computer science can be solved with another level of indirection… Except for the problem of too many layers of indirection" David Wheeler**
- **Goes hand in hand with the layering abstractions**
- **Benefits**
  - Human convenience
  - Makes underlying changes transparent
- **Examples**
  - Host names versus IP addresses

Names versus addresses

- **Names are easier to remember**
- **Addresses can change underneath**
- **Name could map to multiple IP addresses**
  - E.g. load balancing, or geographically closer server
- **Multiple names for the same address**
- **Need a way to map one to the other**
  - DNS hierarchy

Many Translations

- **DHCP: Given a MAC Address, assign an IP address**
  - Uses IP broadcast to find server
- **ARP: Given an IP address, find Ethernet MAC Addresses**
  - Uses Link Layer broadcast to find node
- **DNS: Given a Name, find an IP address**
  - Uses IP unicast/anycast to well known roots, to bootstrap
  - Relies on IP routing infrastructure, DNS hierarchy
- **DHT: Given a key, find a node**
  - Uses IP unicast plus efficient flat namespace routing

Caching

- **Duplicate data stored elsewhere**
  - Reduce latency for accessing the data
  - Reduce the load on other parts of the system
- **Often quite effective**
  - Locality of reference: temporal locality and small set of popular items
- **Examples:**
  - Web caching
  - DNS caching
  - ARP caching
  - Learning bridges

DNS Caching

- **What is cached?**
  - Mapping of names to IP addresses
  - Lookups that failed
  - IP addresses of name servers
- **Reduces latency**
- **Reduces load on hierarchy**
- **Why is it effective?**
  - Mostly read database
  - Doesn't change very often
  - Popular sites are visited often

HTTP Caching

- **What is cached?**
  - Web objects
- **Where is it cached?**
  - Browser, proxy-cache, main memory on server
- **Reduces latency, load**
- **What contributes to high hit rates?**
  - Cacheable content (mostly static)
  - Sharing the cache among multiple users
  - Small amount of popular content

Randomization

- **Distributed adaptive algorithms**
- **Risk of synchronization**
  - Many parties respond to the same conditions in the same way
  - May lead to bad aggregate behavior
- **Randomization can de-synchronize**
  - Example: Ethernet backoff mechanism
  - Example: Random Early Drop

Soft State

- **State is stored in nodes by network protocols**
  - E.g., a mapping, routing entry, cached object
- **Key issue: how to deal with changes?**
- **Hard state: “valid unless told otherwise”**
  - “Managed” by originator of state
  - Kept consistent, explicit invalidation
- **Soft state: “valid if fresh”**
  - Removed by storing node on *timeout*
  - Periodically refreshed as needed
    - May need extra cost (on-demand revalidation or check)
  - Can be seen as a hint
- **Soft state reduces complexity**

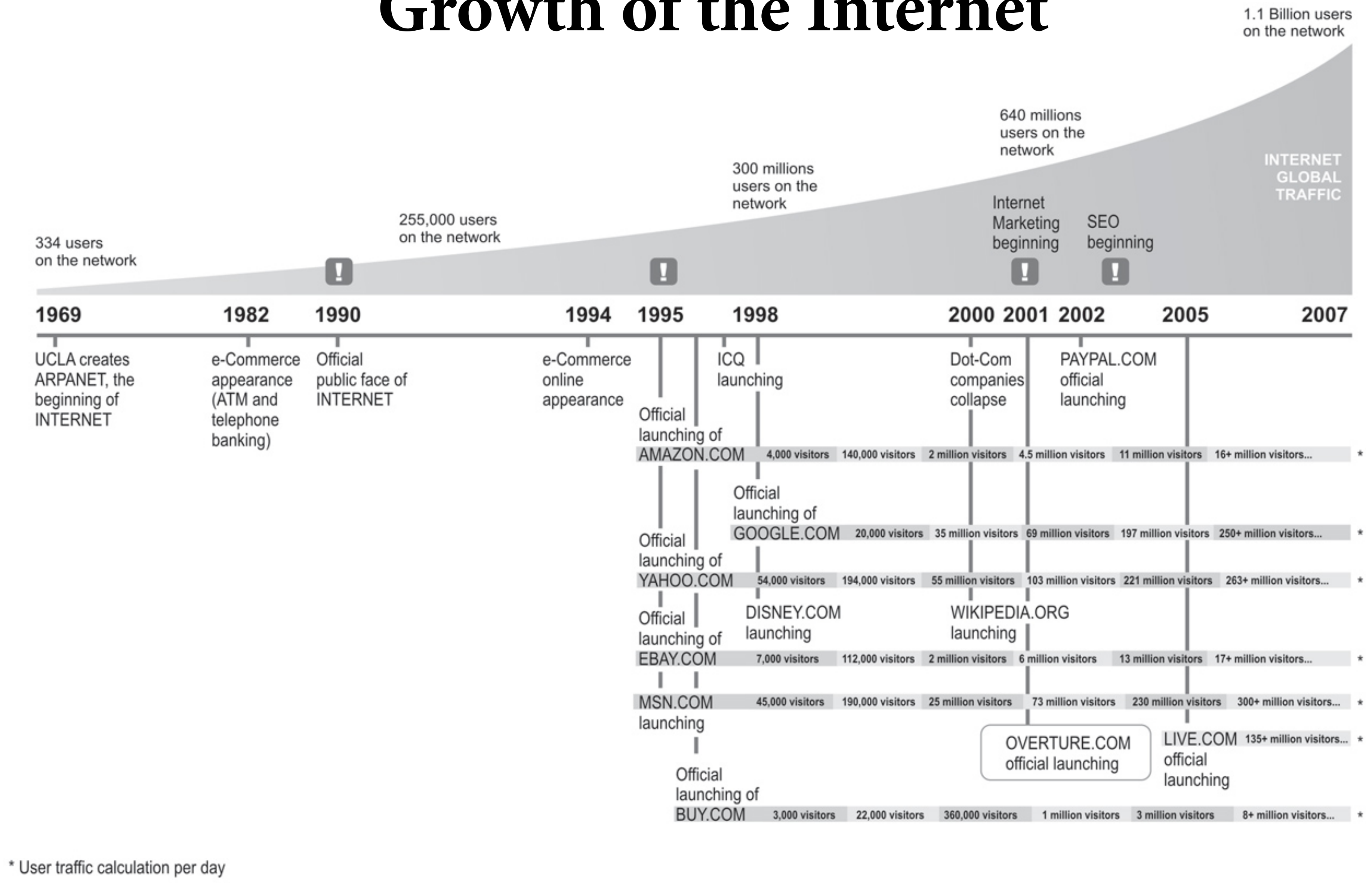
Soft state examples

- **DNS Caching**
  - TTL
  - Can be wrong, check with origin on error
- **Alternative**
  - Origin keeps track of copies
  - Refresh copies on change in mapping
- **Cache coherence is hard**
  - And expensive at scale!
- **Others**
  - DHCP lease

Internet Architecture

- **A Radical Idea**
  - Dumb network
  - Lowest common denominator (best-effort service)
  - No reservations: statistical multiplexing, packets
- **Amazingly successful**
  - Architecture has scaled in size…
  - Many orders of magnitude difference in bandwidth, latency, jitter, reliability, …

Growth of the Internet



Source: Miguel Angel Todaro

Original Design Principles of the Internet

- **David Clark, 1988 "The Design Philosophy of the DARPA Internet Protocols"**
- **Fundamental Goal:**
  - Effective technique for multiplexed utilization of existing interconnected networks
- **Secondary Goals:**
  - Communication should continue despite loss of nodes
  - Multiple types of service
  - Variety of networks
  - Distributed management of resources
  - Cost effective
  - Low-effort host attachment
  - Resources must be accountable

But… There are BIG Challenges

- **Designed in a different environment, with different uses**
  - Identity / Accountability
  - Access model
  - Security
  - Challenges to openness

Identity

"On the Internet, nobody knows you're a dog."

- **Leads to**
  - Spoofing
  - Spam
  - Denial of service
    - Amplification attacks
  - Route hijacking
  - DNS cache poisoning

Protocols designed based on trust

- **That you don't spoof your address**
  - MAC spoofing, IP spoofing, email spoofing
- **That you are who you say you are**
  - BGP announcements, Websites, DNS servers
- **That you adhere to the protocol**
  - Ethernet exponential backoff after a collision
  - TCP-friendliness
- **That protocol specifications are public**
  - So that others can build interoperable implementations

Nobody in charge

- **Traffic traverses many Ass**
  - Who's at fault when things go wrong?
  - How do you upgrade functionality?
- **Anyone can add any application**
  - Whether it is legal, moral, good, well-behaved, etc.
- **Nobody knows how big the Internet is**
- **Spans many countries**
  - So no government can be in charge

Access Models

- **“On by default”**
  - Any node can talk to any node (IP, email, web)
  - Allows for Denial of Service Attacks!
  - Can use a firewall…
    - But won’t stop attackers from saturating the paths to you!

Host versus Data centric

- **IP is host-to-host protocol**

  telnet myhost.mycompany.com

- **Today**
  - Users want content, not servers
  - Web: many redirections, lots of URLs are not "human readable"

    http://a7.sphotos.ak.fbcdn.net/hphotos-ak-ash1/167898_788691982781_7555_40937029_2012165_n.jpg
  - "Lookup" through search engines
  - BitTorrent: torrent file describes content, specific peers are irrelevant
- **Can the architecture support this better?**

Security

- **Last class**
- **Huge challenges**
  - Public Key Infrastructure
  - S-BGP, DNSSEC, IPSec
- **Spoofing, Poisoning, Phishing**
- **Denial of Service attacks**
- **Cyber-security**
  - Cyber-war (talk to John Savage)

Challenges to Openness

- **Walled Gardens**
  - E.g., Facebook, Google
  - Convenient, easy to use, network effects
  - Intrusive data collection
  - No control of own data, hard to migrate
  - Centralization of trust
  - Proprietary protocols
- **Network Neutrality**
  - Should all packets be treated equally?
  - ISPs are commoditized, want to make money
  - Can prioritize own traffic, charge to carry other traffic
  - Very hot debate topic

Other Challenges

- **Extreme mobility**
  - Mobile with no fixed attachment point
  - How to maintain efficient routing?
- **Large number of nodes**
  - Billions of small networked devices (e.g., sensors)
  - “Internet of Things”
- **Sometimes-connected nodes**
  - Developing regions with intermittent connectivity
- **Real-time applications**
  - VoIP, gaming, IPTV

Future of the Internet

- **Can we fix these problems**
  - Security
  - Performance
  - Upgradability
  - Manageability
  - … your favorite ailment here …
- **Without disrupting a critical infrastructure?**

- **Open technical and policy question…**

Thank you!