CSE 461: Introduction to Computer Communications Networks
Winter 2009

Module 2
Overview of Computer Networks

John Zahorjan
zahorjan@cs.washington.edu
534 Allen Center

Today's Topics

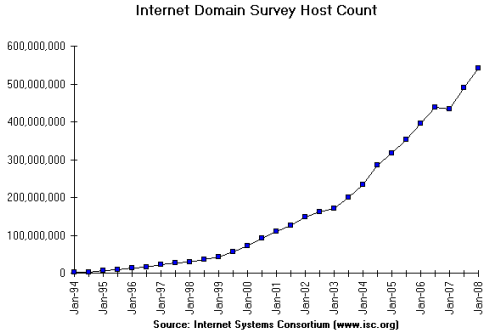
Overview of Computer Networking

1. Scalability / Implications of scale
2. The API
3. Internet overview
4. Layering / The OSI Model

Part 1: Network Scalability

- For this course, a “network” is what connects two or more computers. (What’s a “computer”?)
- We are interested in network architectures that are “scalable” – continue to work efficiently even as the size of the system grows by orders of magnitude

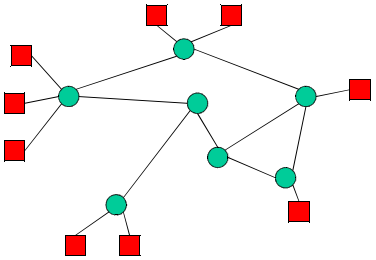
Why is scalability important?



- The basic network design happened in the 1970’s.
  - There were maybe 10,000’s of computers in the world at the time
- Not only *could* the design scale, it provided a combination of cost and benefit that drove demand

Implication of Scale I: Sharing

- It's clearly infeasible to interconnect an ever growing number of machines by running a wire/fiber/radio wave between every pair



- Links carry information (bits)
  - Wire, wireless, fiber optic, smoke signals …
  - May be point-to-point or broadcast
- Switches move bits between links
  - Routers, gateways,bridges, CATV headend, PABXs, …
- Hosts are the communication endpoints
  - PC, PDA, cell phone, tank, toaster, …
  - Hosts have names
- Much other terminology: channels, nodes, intermediate systems, end systems, ...

Implication of Scale II: Intrinsic Unreliability

- Information sent from one place to another
  - May not arrive
  - May arrive more than once
  - May arrive in garbled fashion
  - May arrive out of order
  - May be read by others
  - May be modified by others
- Why build intrinsically unreliable networks?

Implication of Scale III: Distributed

"A distributed system is a system in which I can't do my work because some computer has failed that I've never even heard of." – Lamport

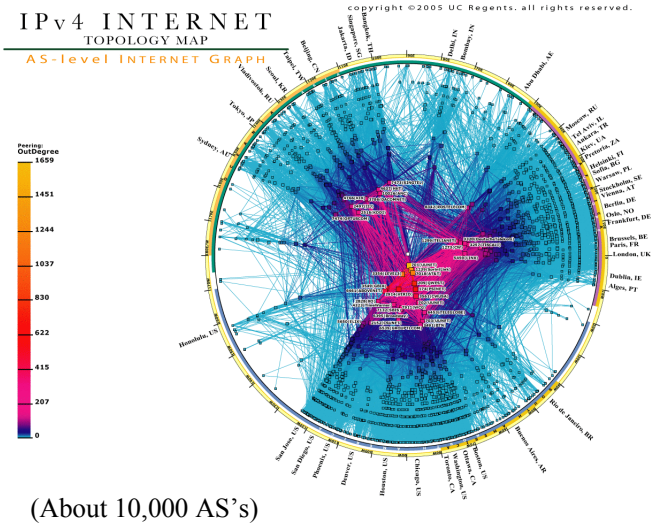
- (Hopefully) independent failure modes
- Exposed and hidden dependencies

Impl. Of Scale IV: Heterogeneous HW/SW

- Heterogeneous: Made up of different kinds of stuff
  - vs Homogeneous: Made up of the same kind of stuff
- Principles
  - Homogeneous networks are easier to deal with
  - Heterogeneous networks promote innovation and scale
  - Consider telephone network vs. Internet
  - Reasons?

Implications of Scale V: Autonomous Authorities

- The Internet is basically an interconnection of networks owned and operated by different people/corporations
  - I own/operate the network in my house
  - My ISP owns the network my network directly connects to
  - My ISP is connected to the network owned by the UW
  - The UW's network is connected to the network owned by CSE



(About 10,000 AS's)

Implications of Confederation/Autonomy

- HW/SW heterogeneity (which was inevitable due to scale anyway)
- "Okay everybody, start using 64-bit addresses NOW."
- Policy/goals heterogeneity
  - So what?

Summer 2006: Internet Neutrality

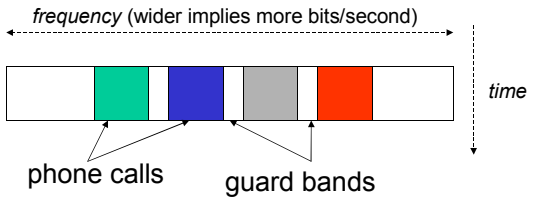
2007: Cell Service Neutrality

How to share: multiplexing

- How should you multiplex (share) a resource among multiple users?
  - In particular, how do you share network links and switches?
- There are two classes of approaches:
  - Static Partitioning ("Reservations")
    - Time Division Multiplexing (TDM)
    - (Space) Frequency Division Multiplexing (FDM)
  - Statistical Multiplexing ("On demand")
    - Packet Switching

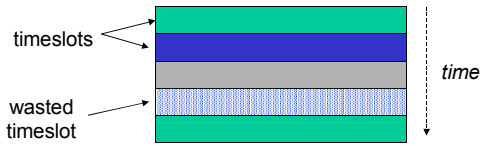
Frequency Division Multiplexing

- Simultaneous transmission in different frequency bands
- "Speaking at different pitches"
  - e.g., take one 3MHz signal and break it into 1000 3KHz signals
    - Analog: Radio, TV, AMPS cell phones (800MHz)
  - also called Wavelength DMA (WDMA) for fiber



Time Division Multiplexing

- "Slice up" the single frequency band among users
- "Speaking at different times"
  - Digital: used extensively inside the telephone network
  - T1 (1.5Mbps) is 24 x 8 bits/125us; also E1 (2Mbps, 32 slots)



Statistical Multiplexing: Packet Switching

The basic idea is very familiar from everyday life (e.g., washrooms on airplanes).

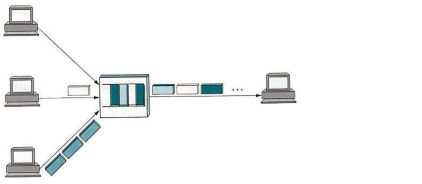


Figure 1.6 A switch multiplexing packets from multiple sources onto one shared link.

Bandwidth is allocated on demand.

Statistical Multiplexing

- Three "details":
  - why is there buffering?
  - why must there be a maximum packet size (even if there were infinite buffering)?
  - under what conditions might a switch run out of buffers?
- Static multiplexing can suffer large overheads when some client can't make use of its allocation.

  Are there any overheads involved in packet switching?

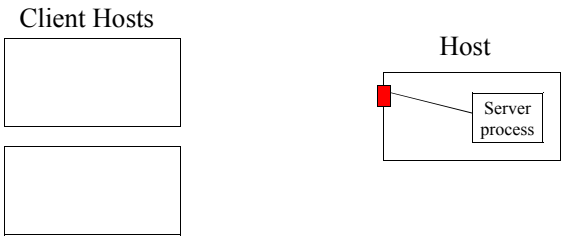
Statistical vs. Static / Performance Measures

- Which is better?
  - We have to decide what we mean by better.
  - We often do this by talking about types of performance measures, and the kinds of workloads that care about them.
    (This gets us near quality-of-service issues, which are addressed late in the course.)
- There are many different performance measures one might be interested in
  - average throughput (goodput)
    - important when you're sending a lot of data (e.g., file transfer)
  - average latency
    - important when you're sending a little data and you want a response (telnet/ssh)
  - variance in throughput and latency (jitter)
    - important to streaming media (audio, video, Skype (VoIP))
    - real time systems (e.g., airplane flight control)
  - minimum guaranteed throughput / maximum guaranteed latency
    - when does the client know that it won't get it's minimum?
    - example use: deciding on an encoding quality for streaming audio/video
- For which is statistical better, and for which static?
- The Internet uses (primarily) statistical mpx. Why do all the kinds of applications above run on it? *(Okay, maybe not airplane flight controls…)*

Part II: The API

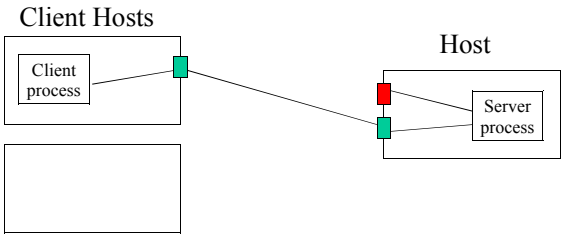
- Just as we want the network service software to run on top of many kinds of hardware, we'd like many kinds of applications to run on top of the network service
- The API is most commonly exposed through a *socket interface*
- A socket is a communication endpoint

Socket API – The Typical Case



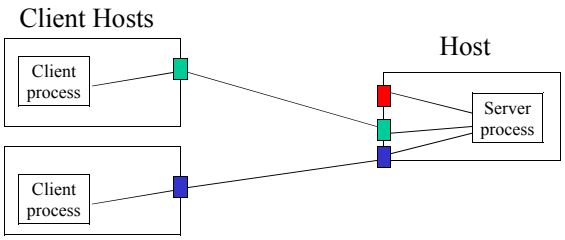
1. Server process is launched, creates a socket, and waits someone to connect to it.

Socket API (2)



1. Server process is launched, creates a socket, and waits someone to connect to it.
2. Client process is launched on some host, creates a socket, and causes it to be contact the server-side socket. This creates a new socket at the server, representing this particular connection.

Socket API (3)



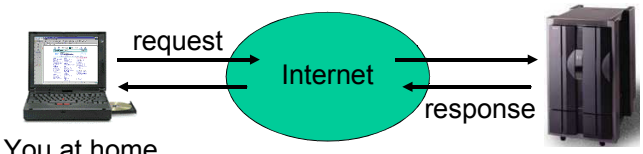
1. Server process is launched, creates a socket, and waits someone to connect to it.
2. Client process is launched on some host, creates a socket, and causes it to be contact the server-side socket. This creates a new socket at the server, representing this particular connection.
3. Another client does the same thing…

Socket API

- Somewhat more detail in Chapter 1
- Somewhat more detail as part of doing HW 2
- What value is there to putting sockets between processes? (Why not connect to the server process directly?)

Part III: A Brief Tour of the Internet

- What happens when you "click" on a web link?



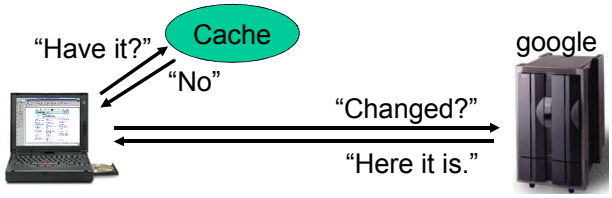
You at home
(client)

www.google.com
(server)

- This is the view from 10,000 ft …

9,000 ft: Scalability

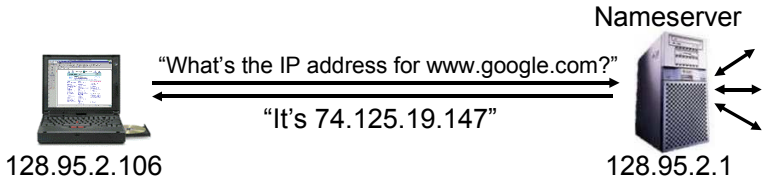
- Caching improves scalability



- We cut down on transfers:
  - Check cache (local or proxy) for a copy
  - Check with server for a new version

8,000 ft: Naming (DNS)

- Map domain names to IP network addresses



- All messages are sent using IP addresses
  - So we have to translate names to addresses first
  - But we cache translations to avoid doing it next time

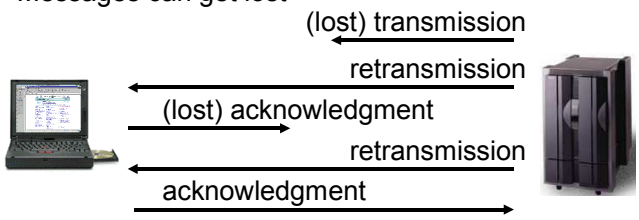
7,000 ft: Sessions (HTTP)

- A single web page can be multiple "objects"

- Fetch each "object"
  - either sequentially or in parallel

6,000 ft: Reliability (TCP)

- Messages can get lost



- We acknowledge successful receipt and detect and retransmit lost messages (e.g., timeouts)

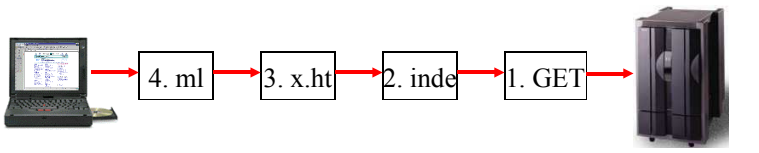
5,000 ft: Congestion (TCP)

- Need to allocate bandwidth among users

- Senders balance available and required bandwidths by probing network path and observing the response

4,000 ft: Packets (TCP/IP)

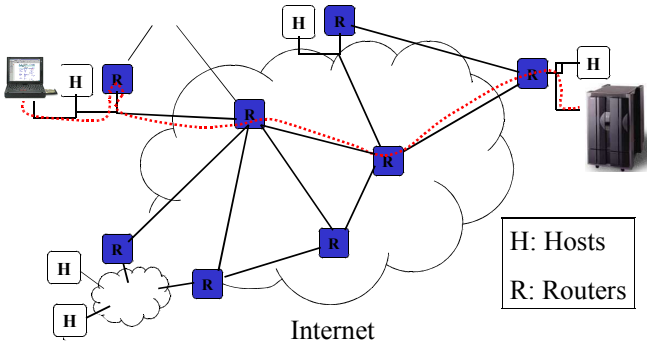
- Long messages are broken into packets
  - Maximum Ethernet packet is 1.5 Kbytes
  - Typical web page is 10 Kbytes



- Number the segments for reassembly

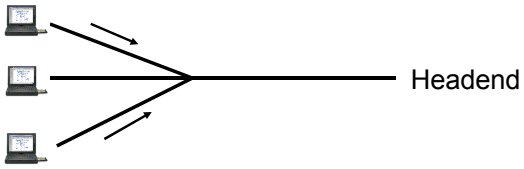
3,000 ft: Routing (IP)

- Packets are directed through many routers



2,000 ft: Multi-access (e.g., Cable)

- May need to share links with other senders



- Poll headend to receive a timeslot to send upstream
  - Headend controls all downstream transmissions
  - A lower level of addressing (than IP addresses) is used … why?

1,000 ft: Framing/Modulation

- Protect, delimit and modulate payload as signal

Sync / Unique	Header	Payload w/ error correcting code

- E.g, for cable, take payload, add error protection (Reed-Solomon), header and framing, then turn into a signal
  - Modulate data to assigned channel and time (upstream)
  - Downstream, 6 MHz (~30 Mbps), Upstream ~2 MHz (~3 Mbps)

Part 3. Protocols and Layering

- We need abstractions to handle all this system complexity

  A protocol is an agreement dictating the form and function of data exchanged between parties to effect communication

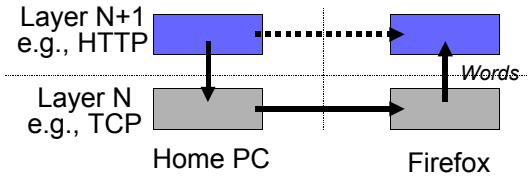
- Two parts:
  - Syntax: format -- where the bits go
  - Semantics: meaning -- what the words mean, what to do with them
- Examples:
  - Ordering food from a drive-through window
  - TCP/IP, the Internet protocol
  - HTTP, for the Web

Protocol Standards

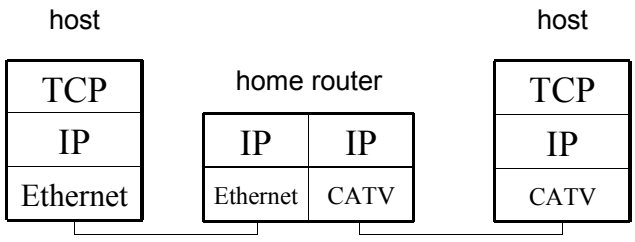
- Different functions require different protocols
- Thus there are many, many protocol standards
  - E.g., IP, TCP, UDP, HTTP, DNS, FTP, SMTP, NNTP, ARP, Ethernet/802.3, 802.11, RIP, OSPF, 802.1D, NFS, ICMP, IGMP, DVMRP, IPSEC, PIM-SM, BGP, …
  - every distributed application requires a protocol…
- Organizations: IETF, IEEE, ITU
- IETF (www.ietf.org) specifies Internet-related protocols
  - RFCs (Requests for Comments)
  - "We reject kings, presidents and voting. We believe in rough consensus and running code." – Dave Clark.

Layering and Protocol Stacks

- Layering is how we combine protocols
  - Higher level protocols build on services provided by lower levels
  - Peer layers communicate with each other

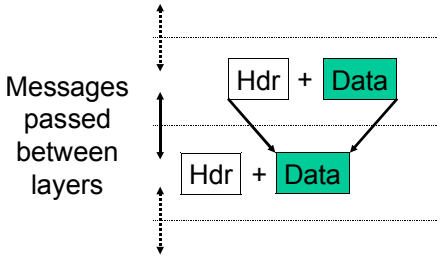


Example – Layering at work



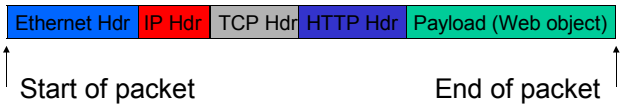
Layering Mechanics

- Encapsulation and de(en)capsulation



A Packet on the Wire

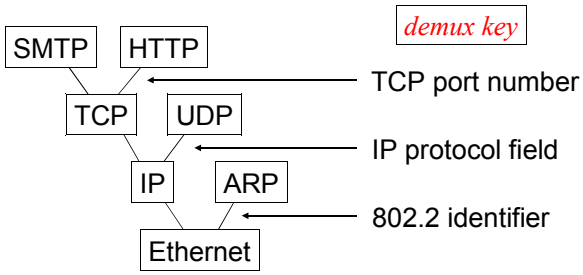
- Starts looking like an onion!



- This isn't entirely accurate
  - ignores segmentation and reassembly, Ethernet trailers, etc.
- But you can see that:
  - layering adds overhead
  - one protocol's header is another protocol's data

More Layering Mechanics

- Multiplexing and demultiplexing in a protocol graph



Part 4. OSI/Internet Protocol Stacks

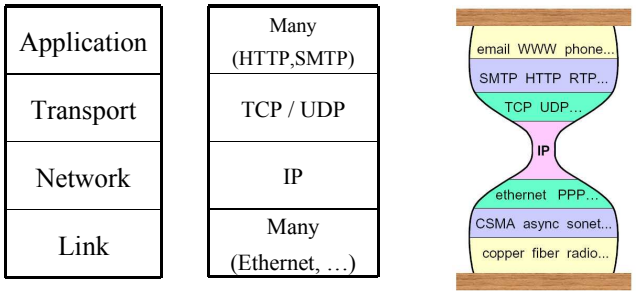
Key Question: What functionality goes in which protocol?

- The "End to End Argument" (Reed, Saltzer, Clark, 1984):

  Functionality should be implemented at a lower layer only if it can be correctly and completely implemented. (Sometimes an incomplete implementation can be useful as a performance optimization.)

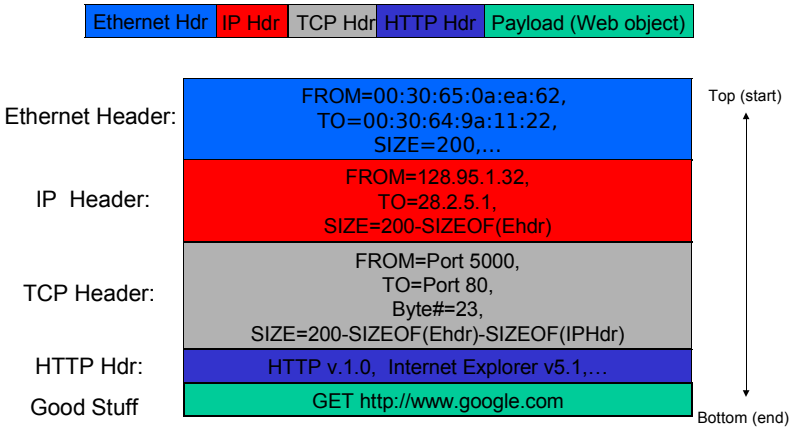
- Tends to push functions to the endpoints, which has aided the transparency and extensibility of the Internet.

Internet Protocol Framework

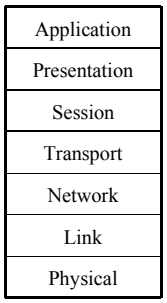


Model Protocols The “narrow waist”

What’s Inside a Packet



OSI “Seven Layer” Reference Model



Their functions:

- Up to the application
- Encode/decode messages
- Manage connections
- Reliability, congestion control
- Routing
- Framing, multiple access
- Symbol coding, modulation