CS 140 – Summer 2008 – Lecture #22: Networking Overview, Link Layer

Next topic: Networking

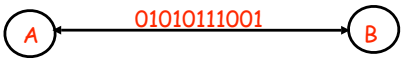
- Communication:
  - **THE* killer application of computer (and operating?)**
  - **systems (entertainment: close second?)**
- So, mini-course on Networking, at no extra cost!
  - **for the real deal, take CS144/244A/B/C**
- Starting with:
  - **communication as the ultimate social solvent**
  - **the magical properties of symbolic information**
  - **some fundamental issues in moving it around**
- Readings:
  - **Ch. 16 (skim)**
  - **Butler Lampson's networking notes (skim the actual code)**

The power of communication

- Communication very strange topic
  - **Sounds trivial, but is an incredible force.**
  - **New ways to transmit/represent information = revolution.**
- Examples:
  - **Books: enables communication through time, across geography. Changed world.**
  - **Voice: communicate over air. Changed the planet.**
  - **Telephone: escape tyranny of geography: talk to anyone in world. Try to find someone that hasn't used it.**
  - **TV: revolutionary implication: watch CS140 in bed.**
- Can view communication as transport
  - **(moving signal over geography)**
  - **New transport = revolution. Ships, rail, cars, planes, …**

The power of computer networks

- Computer networks:
  - **Send symbolic information (1's and 0's) between "nodes".**



- A universal substrate, same plumbing supports many revolutions:
  - **Email/IM: talk to anyone in world in a different way.**
  - **News/bulletin boards: talk to bunch of different people.**
  - **Web: made grandma use computer.**
  - **The implications of these still playing out…**
  - **(note: these are all basically *file transfer*)**

Computer networks carry symbols

- The history of transport dominated by moving things:
  - **Trucks, railroads, cars to move people, goods, etc**
- Networks are different: move bits/files
  - **The great invention: the packet. Encapsulates information in an opaque sequence of 1s and 0s. (Container ships?)**

  - **Result: totally general, can carry any sequence of bits**
- Information is extremely flexible/mungible:
  - **Recursive: wrap packet in another packet**
  - **Partitionable: split packet arbitrarily and then reassemble**
  - **Time independent: information can flow at different rates**
  - **Encoding independ': translate into another form and back**
  - **Generic: completely general error detection/correction**

Some example networks

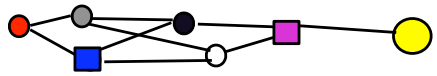
- ARPAnet:
  - **First widely-used networking, developed in early 1970s but still in use. Connected together large timesharing systems all over country using leased phone lines.**
  - **Provided mail, file transfer, remote login**
- Usenet:
  - **Developed late 70s early 80s. Unix systems phone each other up to send mail and transfer files**
- Local area networks (LANs)
  - **Developed early 80s to hook together workstations. Most popular is Ethernet. LANs very different from WANs.**
  - **Late 90s/early 2000s: WLANs take off.**
- Internetworks:
  - **Mechanisms for tying together existing networks.**

Telephony: a parallel networking universe?

- 1890s onward: wired analog phone network
- 1920s: teletypewriters
- 1970s: radio phones
- 1980s: digital phone network
- 1980s-present: cellular telephones
- 1990s-2000s: text messaging, smart phones...
- 2000s: IP telephony (VoIP)
- Convergence with computer networks?
  - **Carrier lock-in, non-interoperability, but there are signs: Verizon, Skype, iPhone...**
- Cable/TV networks?

Networking's Big Win: Decentralization

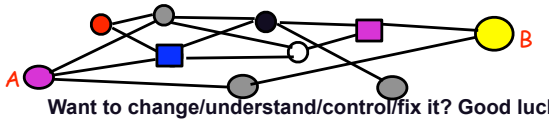
- Distributed, parallel, unsynchronized growth/operation!
  - **Rather than one big thing, build out of pieces and connect**



  - **Result: the biggest things in the world are network-based ("distributed systems"): Internet, telephone system…**
- Decentralization gives:
  - **Robustness: one part breaks, who cares.**
  - **Piecemeal construction (rather than all at once).**
  - **Modularity: end point internals don't matter: just adhere to protocol and can talk.**
  - **Massive concurrency: each node doing its own thing.**

Networking's Big Failure: Decentralization

- A giant pile of unmanageable stuff
  - **Rather than one big thing, many little pieces.**

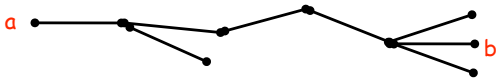


  - **Want to change/understand/control/fix it? Good luck!**
- Decentralization/distributed system gives us:
  - **Failure: many parts = (always) some broken parts.**
  - **Piecemeal construction: have no idea what net looks like.**
  - **Problem: How to get from a to b?? How to even name?**
  - **Heterogeneity: Have to talk over many different technologies.**
  - **Massive concurrency = massive complexity.**

Our big solution: layers.

- We'll look at networks as made of three main layers:
  - **Link level: physically encode bits on "wire" (line segment).**

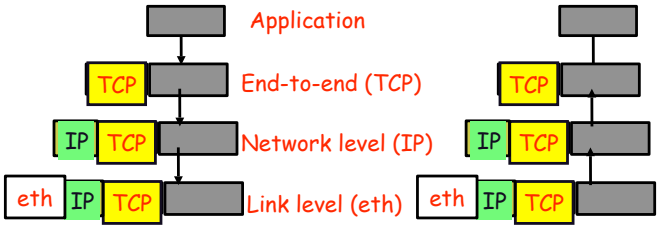
  - **Network layer: connecting segments, addressing (locating points on graph) and routing (navigating graph).**



  - **End-to-end layer: making network simple and reliable.**

How do layers work?

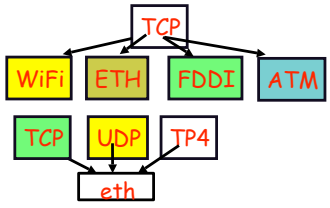
- Rules:
  - **Layers do not look inside packet.**
  - **If they need auxiliary information, attach a header to message on way down, strip on way up.**



  - **Protocol = Contract to talk to others at same level.**

Why layers?

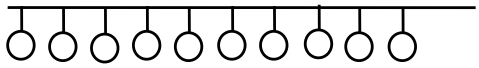
- Major reason: independence
  - **Layers treat each other's headers as opaque.**
  - **Change lower without changing upper.**
  - **Change upper without changing lower.**



- Where?
  - **Evolve at different rates? stick layer between them.**
  - **Functionality needed by ~ all clients? move down.**
  - **Functionality not needed by all clients, move up.**

Some networking fundamentals

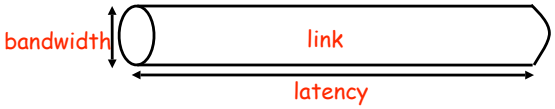
- Networking connects geographically separated things
- Implication 1: speed of light important
  - **Only 30cm / nanosecond!**
  - **Can't get better.** SF — 15 milliseconds → Boston
- Implication 2: sharing (multiplexing)
  - **Laying pipe very expensive:**
    - **~10 billion to replace wire with fiber in Britain.**
    - **Slice of radio spectrum cost 1.7 billion at FCC auction.**
  - **Amortize cost by putting multiple machines on each link.**



  - **Multiple users, one resource = need to multiplex**
  - **We've had sharing before. What is new? Blindness.**

Performance: latency & bandwidth

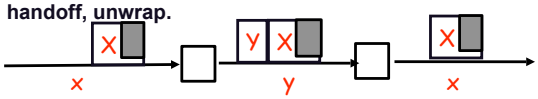
- Latency: how long minimum communication takes
- Bandwidth: number of bits per time unit



- The usual ways to minimize latency?
  - **Caching *reduces* latency when cache hits.**
  - **Prefetching *hides* latency.**
  - **Concurrency *tolerates* latency by doing something else.**
- Save bandwidth?
  - **Which latency trick(s) save bandwidth?**
  - **How to trade CPU for bandwidth?**

Theme 0: recursion

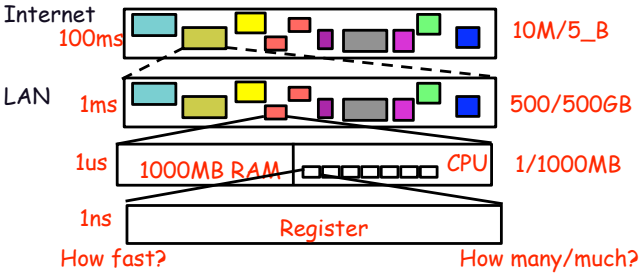
- Construction:
  - **Base case: link**
  - **Inductive step: connect links (or networks)**
- Use: encapsulate packet in another, in another, …
  - **Universal transport: send message over any network.**
  - **Encapsulation: send message using another layer: wrap up, handoff, unwrap.**



  - **Big use: send new protocols across old routers.**
- Presence: each layer in computer system has network
  - **CPU connected to disk by I/O bus, cache connected to memory by memory bus, CPU to register by wire…**

Recursion: it's networks all the way down

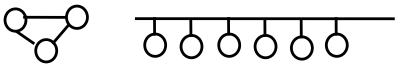
- Every layer in computer has three components
  - **Computation, memory, network to connect.**



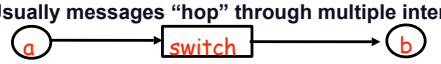
- And up! The social mimics the technical: Network

Theme 1: variations on connectivity

- Point-to-point vs. one-to-all (broadcast) connectivity



  - **Broadcast medium cheap, but contention a problem.**
- Indirect connectivity:
  - **Rarely have point-to-point connection with destination.**
  - **Usually messages "hop" through multiple intermediaries.**



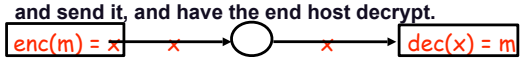
  - **(router: connect one network to another (= internetwork)**
- No connectivity: must deal with failure constantly
  - **Networks tend to be big.**
  - **As n increases, # of dead things does too (link down).**
  - **As n increases, # of overloaded things does too (msg lost).**

Theme 2: synthesized reliability

- Networks are unreliable.
  - **Loses, corrupts, reorders and duplicates messages.**
  - **False premise: must make lower levels perfect.**
- General approach: detect + retransmission
  - **General trick: something dead doesn't respond. So wait "reasonable amount of time", if nothing, assume dead.**
  - **Reliability from lost messages: Send message, wait for acknowledgement, if timeout, re-send. Repeat as needed.**
  - **Reliability from corruption: use checksum to detect flipped bits. Discard if doesn't match.**
  - **Duplication in time, rather than duplication in space.**

The end-to-end argument

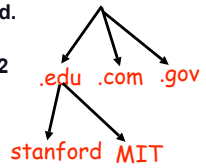
- "Functions at low level may be redundant or of little value when compared to the cost of providing them"
  - **Put in low level: everyone must pay for it.**
- example: useless security
  - **Safely send credit card #: need to encrypt on my machine, and send it, and have the end host decrypt.**



  - **Having my LAN also do encryption useless (only 1 link).**
- Example: harmful reliability (e.g. classic Ethernet, WiFi!)
  - **Links try to synthesize reliability using retransmission**
  - **Introduces delay (send. Wait. Retrans. Repeat).**
  - **If traffic delay sensitive (voice, video) this is exactly the wrong thing to do! better to lose packets and continue.**

Theme 3: addresses (i.e. naming)

- Connect many things? need way to name them
  - **Addresses show up at each level.**
- Flat address:
  - **No structure, usually fixed sized.**
  - **Only operation = equality.**
  - **Ethernet address 8:0:2b:e4:b1:2**
- Hierarchical address:
  - **Tree of flat addresses.**
  - **Usually variable sized.**



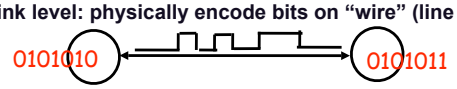
If one address a prefix of another, shorter address "contains" (is the parent of) the other.

A few address examples

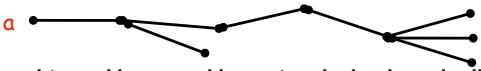
System	address	sample address	data value
LAN/MAC	6 byte flat	ff f3 63 23 a1 92	packet
IP	4 byte hierarchical	16.12.3.134	packet
TCP	IP+port	16.12.3.134/3451	byte stream
file system	path name	/foo/bar	0-4GB
main memory	32-bit flat	0xf33f0000	8,16,32,64 bits

Layered Architecture: The Big Picture

- Divide network into three layers:
  - **link level: physically encode bits on "wire" (line segment)**



  - **network layer: connecting segments, addressing (locating points on graph) and routing (navigating graph)**

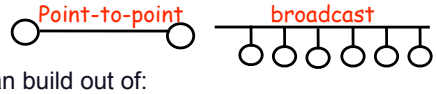


  - **end-to-end layer: making network simple and reliable**

- Summary so far: networking from 50,000 feet.
- Next: Link level (ground zero). Reading: Lampson.

Links

- Link = pipe to send information. Examples:



- Can build out of:
  - **Twisted pair (the wire your phone connects to).**
  - **Coaxial cable (the wire your tv connects to).**
  - **Optical fiber (the stuff we all want to connect to).**
  - **Space (the stuff that does not require laying pipe: voice, radio waves, microwaves, laser beams)**
- What do higher layers require of a link?
  - **Very little: send bits well enough that we don't give up**
  - **assume: links lose, corrupt, reorder, and duplicate pkts.**

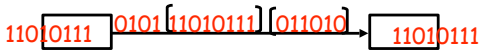
Example links ("b" = bit, "B" = byte)

Medium	link	bandwidth	message
CPU	on-chip bus	24GB/s	8 bytes
pc board	Rambus	3.2GB/s	packet < 100B
	PCIe 2.0	500MB/s	packet
Disk I/O	SATA	100MB/s	packet
LAN	Ethernet	1.25MB/s	packet 64-1500B
	Fast Ethernet	12.5MB/s	packet,64-1500B
	Gigabit Eth.	125MB/s	packet,64-1500B
Wireless	WiFi-N	37.5MB/s	< 1500 B
copper pair	ISDN	64-128Kb/s	1 byte
copper pair	DSL	128-24,000Kb/s	1 byte
copper pair	T1	1.5Mb/s (24 ISDN's)	1 byte
coax cable	T3	44.736Mb/s (30 T1's)	1 byte
coax cable	Cable Modem	256-30,700 Kb/s	< 64K bytes
fiber	STS-1	51.8Mb/s	1 byte or cell
	STS-48	2.5Gb/s (48 STS-1's)	1 byte or cell
	(STS-* also called OC-*)		

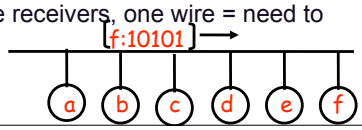
Link level issues

- Encoding: map 0 and 1 down to wire and back

- Framing: delineate bit stream into frames (packets) receiver knows where things begin

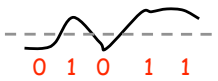


- Arbitration: multiple senders, one resource = need to coordinate
- Addressing: multiple receivers, one wire = need to indicate which one



Encoding

- Goal: map 0's and 1's onto wire and back.
- Simple scheme:
  - **0 = low voltage**
  - **1 = high voltage**
  - **Called "non-return to zero" (NRZ)**
  - **Doesn't work too well.**

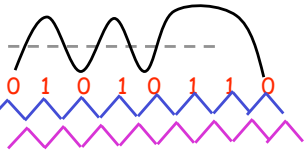


As dumb as the name

- Problem 1: How to tell errors from packets?
  - **What does a dead link look like?**
  - **What does a jammed link look like?**
  - **(fix: map data in such a way that always has transitions)**
- Problem 2: More basic: when should we sample?!

The big sampling problem: Clock drift

- To send bits: every clock cycle
  - **Sender encodes a bit.**
  - **Receiver decodes a bit.**
  - **But: don't have same clock!**

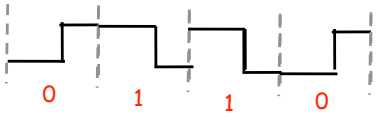


What happens if out of phase?

- Sol'n: Clock recovery: derive clock from data
  - **Insight: whenever signal changes from 0->1 or 1->0 receiver knows it is on a clock boundary.**
  - **So, encode data so that it always has transitions!**
  - **E.g.: Manchester encoding (used by Ethernet).**

Manchester encoding

- Goal: want both 0 and 1 to give a transition
  - **Map 0 to low-high transition.**
  - **Map 1 to high-low transition.**

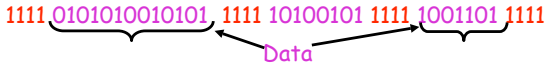


- Good: can detect:
  - **Dead link, short, and collisions (how?)**
- Bad: reduces data rate
  - **If wire can do N transitions per second, what is NRZ's data rate? Manchester encodings?**
  - **Other encoding schemes fight this.**

Could have sent two bits using NRZ for each 1 bit

Framing: splitting bits into packets

- Why packets?
  - **Look for address in stream, know when to stop reading.**
- Simplest approach: sentinel
  - **send "begin packet" and "end packet" tokens.**
  - **e.g., if sentinel is "1111":**



- Main problem: what if sentinel appears in data?
  - **The usual approach: "escape" sentinel (map to something else; receiver will have to map back) "bit stuffing".**
  - **Example: in C, '\' is a special character, if you need it, have to include it twice "\\".**

HDLC (high-level data link control protocol)

- Same sentinel for begin and end: 0111 1110
- packet format:

- bit stuffing
  - **Sender: if data contains five consecutive 1's, insert a 0 after it:**

0111 1110 → 0111 1101 0

  - **Receiver: if data = five 1's followed by a 0, remove 0.**

0111 1101 0 → 0111 1110

  - **Otherwise read next bit. A 0 implies? A 1 implies?**

Arbitration: coordinating multiple senders

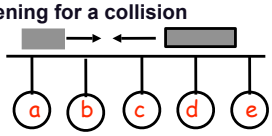
- Problem: one wire, multiple senders
  - **Old problem (CPU, disk, memory). A new thing: no centralized control.**
  - **Many different approaches.**
- Time-division multiplexing (telephone model)
  - **Divide into time slices and round robin among senders.**
  - **Exceed capacity? Don't admit new senders (busy signal).**
- Frequency division multiplexing (radio/TV model)
  - **Divide spectrum up into slices; give each sender a piece.**
  - **Capacity exceeded? Don't give any more slices .**
- Both give fixed delays and constant data rate
  - **Good for continuous data streams.**
  - **Doesn't work well for computer (bursty) traffic.**

Statistical multiplexing

- Two problems with TDM and FDM
  - **Does not adapt. Sender has no data? Slot wasted. Lots of data? Can only send as much as slot allows.**
  - **Need to know maximum senders ahead of time.**
  - **(if net = memory, what do TDM/FDM correspond to?)**
- Two key observations:
  - **Links usually idle & computer network traffic bursty.**
- Two key ideas:
  - **Idea 1: transmit on demand (when sender has data)**
  - **result: link not wasted, get peak bandwidth**
  - **Idea 2: upper bound on packet that can be sent.**
  - **Result? (hint: view packet as process)**
- Problem: "congestion" (too many senders ready at the same time)

Classic Ethernet

- "Carrier Sense, Multiple Access with Collision Detect"
  - **Statistical multiplexing: nodes send when they want.**
  - **Carrier sense = can distinguish between idle and busy.**
  - **Collision detect = if your packet hits another, can hear it.**
- To send data, node:
  - **Listens until link idle**
  - **Immediately sends, while listening for a collision**
- To handle collision:
  - **Jam wire so all senders hear.**



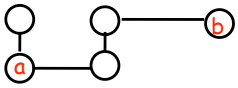
  - **When to retransmit?**
  - **Wait until idle and retry? But will hit again!**

Exponential backoff

- Pretty idea (from Aloha radio network):
  - **Pick a random number distributed between [0..n).**
  - **Wait this amount. Then retransmit.**
  - **If collide again, pick a number between [0..2n), …**
  - **Result: coordination without centralized control!**
- Nice effects:
  - **Adaptive: wait determined by how busy wire is.**
  - **If really busy, everyone will back off until they reach a point that contention goes away.**
  - **Exponential back off used in lots of places to handle contention without some central control (locks, TCP).**
  - **Note: doing retransmit at this level is just an optimization…**

Addressing

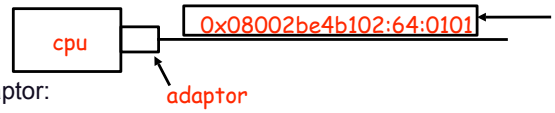
- Problem: how to send from a to b?
  - **If point-to-point, trivial (only one possible receiver)**
  - **How to handle if multiple listeners?**
- On a mesh:
  - **Uniquely name nodes (how?)**
  - **Sender must know name (how?)**



  - **Nodes that join links ("switches" or "routers" depending on level) have to know if & where to forward pkts.**
  - **We'll examine this more when we look at IP.**
- On a broadcast network, solution is simple:
  - **Also give unique names.**
  - **Sender appends name to the beginning of each packet.**
  - **Receiver checks each packet for its name.**

Ethernet receiving example

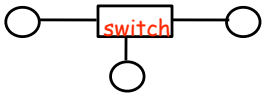
- Node connected to wire by Ethernet adaptor:



- adaptor:
  - **Has unique 6-byte address (e.g., 8:0:2b:e4:b1:2) put in ROM by manufacturer. Can (possibly) change it.**
  - **Scans for packets with its address or with broadcast address (ff:ff:ff:ff:ff:ff)**
  - **When it finds a packet of interest, it buffers it and wakes up CPU.**
  - **Can view this as an example of moving computation to reduce bandwidth. (Otherwise CPU has to scan.)**
  - **Can also view as using parallelism to make system faster.**

Experience with Ethernet

- Work best under lightly loaded conditions
  - **Over 30% and degrades severely due to contention.**
  - **Not much of a problem in practice.**
- Why it works:
  - **Ethernet first viewed as broadcast with lots of nodes, Frequently more like a point-to-point!**



  - **Higher level protocols implement "flow control" which prevents a single node from pounding on network.**
- Why it succeeded: simple to administer + dirt cheap

Bridges: connecting links

- Bridge:

  - **Place between two links.**
  - **Put network adaptor in "promiscuous" mode (see all pkts) whenever receives packet, forwards to the other link.**
- Learning bridge:
  - **Rather than forward all packets, just forward those that are on other side.**
  - **How to figure out? Look at source address in packet!**
  - **Tells you who on receiving side.**
  - **Now when get packet: if destination on sending side, ignore, otherwise forward packet**
  - **(Since network changes, throw out entries over time)**
- Point-to-point links? "Switch."

Link layer summary

- Links carry signals: have to encode bits on these.
- Given a sequence of 0's and 1's, have to break into packets (frames). Usually done using sentinels.
- Multiple senders? Need to coordinate them.
  - **Statistical multiplexing allows you to take advantage of spasmodic traffic.**
  - **exponential backoff counters contention.**
- Multiple receivers? Need to name them.
- Next: the network layer
  - **Issues in finding name of who you want to send to**
  - **routing packet to them.**