

CS 140: Operating Systems Lecture 23: Networking

Mendel Rosenblum

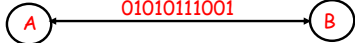
This week: networking

- ◆ Today:
communication as the ultimate social solvent
the magical properties of symbolic information
some fundamental issues in moving it around
- ◆ Readings:
Book (skim)
Kaashoek networking notes (ignore MIT-specific info)
Butler Lampson's networking notes (skim the actual code)


The power of communication

- ◆ Communication very strange topic
Sounds banal, 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: talk to anyone in world in a different way.
News: talk to bunch of different people.
Web: made grandma use computer.
The implications of these still playing out...

Computer networks carry symbols

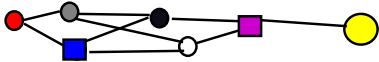
- ◆ The history of transport dominated by moving things:
Trucks, railroads, cars to move people, goods, etc
- ◆ Networks are different: move bits
The great invention: the packet. Encapsulates information in an opaque sequence of 1s and 0s.
Client view: tell A "Hello"  Network view
- Result: totally general, can carry any sequence of bits
- ◆ Information is weird:
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
- ◆ Internetworks:
Mechanisms for tying together existing networks.

A big win of networking

- ◆ Allows decentralization (yet still share)
Rather than one big thing, build out of pieces and connect

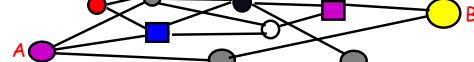


Result: the biggest things in the world are network-based ("distributed systems"): Internet, telephone system...

- ◆ 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.

The big networking problem

- ◆ Forces decentralization
Rather than one big thing, many little pieces.

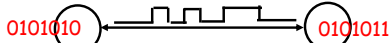


One implication: Cannot change endpoints. Ever.

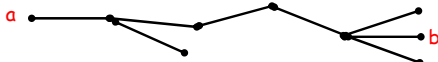
- ◆ Decentralization causes:
 - Failure: many parts = some broken.
 - Piecemeal construction: have no idea what net looks like.
 - Problem: How to get from a to b?? How to even name?
 - Heterogeneity: Have to talk over many different technologies.
 - Massive concurrency = massive complexity.

Our big solution: layers.

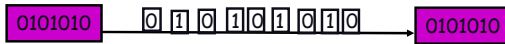
- ◆ We'll look at networks as made of three main layers:
Link level: physically encode bits on "wire" (line segment).



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

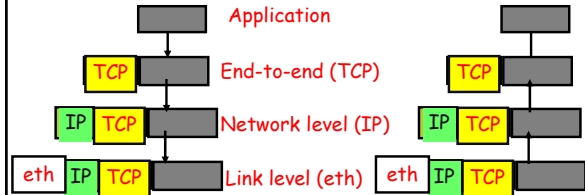


End-to-end layer: making network simple and reliable.



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.

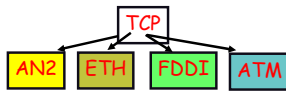


Protocol = Contract to talk to others at same level.

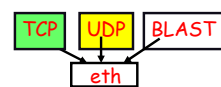
Why layers?

- ◆ Major reason: independence
Layers treat each other's headers as opaque.

Change lower without changing upper.



Change upper without changing lower.

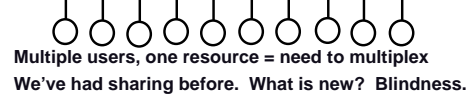


- ◆ Where?
 - Evolve at different rates? stick layer between them.
 - Functionality needed by ~ all clients? move down.
 - Functionality not needed by all clients, move up.

Some networking fundamentals

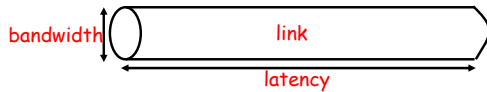
- ◆ Networking connects geographically separated things
- ◆ Implication 1: speed of light important
Only 30cm / nanosecond!
Can't get better. SF $\xrightarrow{15 \text{ milliseconds}}$ Boston

- ◆ 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.



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?
 - Save both: change representation

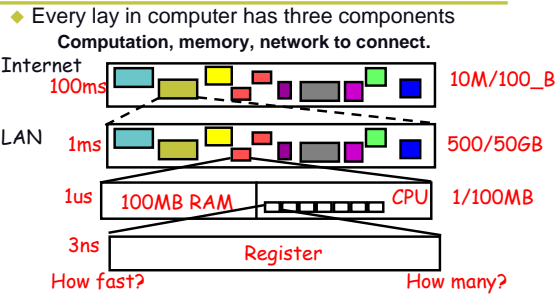
Theme 0: recursion

- ◆ Construction:
 - Base case: link
 - Inductive step: connect links (or networks)
- ◆ Use: encapsulate pkt 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...

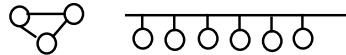
Recursion: it's networks all the way down



- ◆ And up! The social mimics the technical: Network standards bodies mimic network itself.

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.
- ◆ No connectivity: must deal with failure constantly
 - Networks tend to be big.
 - As n increases, # of things does to (link down).
 - As n increases, # of overloaded things does to (msg lost).



Theme 2: synthesized reliability

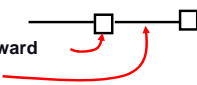
- ◆ Networks are unreliable.
 - Loses, corrupts, reorders and duplicates messages.
 - False premise: must make lower levels perfect.
- ◆ General approach: detect + retransmission
 - General trick: something dead doesn't respond. So wait "reasonable amount of time", if nothing, assume dead.
- Reliability 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.
 - enc(m) = x → x → dec(x) = m
 - Having my LAN also do encryption useless (only 1 link).
- ◆ Example: harmful reliability
 - 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.

The duality of memory and wires

- ◆ Every layer of computer made of three components
Computation, storage, communication
- ◆ The latter two are the same thing:
Storage: communicate through time (multiplexed in space)
Wires: communicate through space (multiplexed in time)
- ◆ Recursively implemented in terms of each other
Storage: use wire to get data from main memory into cache into register into CPU
Wire: use memory to hold message for sending, and to catch message while receiving
- ◆ Connected in terms of each other
Wires: Use a node to buffer and forward
Memories: Use a wire to connect

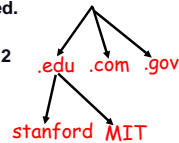


The duality of memory and wires II

- ◆ Reliability: both use duplication
- ◆ Duplication in time:
wires = timeout + resend (using memory)
memory = checkpoint + restore from backup (using wire)
- ◆ Duplicate in space:
wires = send on multiple paths (duplicate wires)
memory = multiple copies (duplicate memory)
- ◆ General naming issues are the same:
name spaces, mapping names to addresses using table, using address to find object
"foo/bar" to a disk block using directories and inodes
leland.stanford.edu to ip address to ethernet address using various name tables

Theme 3: addresses

- ◆ 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	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

Communication: Voice

- ◆ Low-bandwidth broadcast channel.
~18B/sec ("I can say this in about two seconds")
- ◆ Transmit to specific destination?
Name: "Father, please fetch me some Perrier."
Generic name: "You" plus volume ("Hey" prefix optional)
How to get name? Ask someone (analogue: name server)
- ◆ Problems with broadcast medium?
How congestion dealt with?
How is noise (corruption) dealt with?
- ◆ Multiplexing:
FDM: different lecture rooms, same time
TDM: different lecture times, same room

Next time

- ◆ Networking:
Use links to connect nodes.
Use link-level protocol to send data over link.
Use network-level protocol to send data over multiple links.
Use end-to-end protocol to make network seem perfect.