



Internet Design: Goals and Principles

EE122 Fall 2012

Scott Shenker

<http://inst.eecs.berkeley.edu/~ee122/>

Materials with thanks to Jennifer Rexford, Ion Stoica, Vern Paxson
and other colleagues at Princeton and UC Berkeley

Administrivia

- New Office Hours:
 - Thursday 12:00-1:00 in 415 **or** 420 (come find me!)
 - After class on Tuesdays: walk with me to Soda
- Class going more slowly than anticipated
 - Will pivot to real material, skipping some nonessentials
- Lecture on September 18
 - Will be returning from Moscow that day
- Homework #1 released (note submission process)
 - Due in two weeks....this should **not** be hard
 - Project #1 will follow shortly
 - Guesses about dates for future assignments now online

Outline

- Design Goals
- Modularity
- Layering
- End-to-End Principle
- Fate-Sharing

Internet Design Goals

David Clark

- Wrote a paper in 1988 that tried to capture why the Internet turned out as it did
- In particular, it described an ordered list of priorities that informed the design
- We have him with us here today....Eastwood-style

Internet Design Goals (Clark '88)

- **Connect existing networks**
- Robust in face of failures
- Support multiple types of delivery services
- Accommodate a variety of networks
- Allow distributed management
- Easy host attachment
- Cost effective
- Allow resource accountability

Connect Existing Networks

- Wanted single protocol that could be used to connect any pair of (existing) networks
- The Internet Protocol (IP) is that unifying protocol
 - All (existing) networks must be able to implement it
- This is where the need for best effort arose....

Robust

- As long as network is not partitioned, two hosts should be able to communicate (eventually)
- Failures (excepting network partition) should not interfere with endpoint semantics
- *Very successful, not clear how relevant now*
 - Availability more important than recovering from disaster
- *Second notion of robustness is underappreciated*
 - Key to modularity of Internet

Types of Delivery Services

- Use of the term “delivery services” already implied an application-neutral network
- Built lowest common denominator service
 - Allow end-based protocols to provide better service
 - For instance, turn unreliable service into reliable service
- Example: recognition that TCP wasn't needed (or wanted) by some applications
 - Separated TCP from IP, and introduced UDP

Variety of Networks

- Incredibly successful!
 - Minimal requirements on networks
 - No need for reliability, in-order, fixed size packets, etc.
 - A result of aiming for lowest common denominator
- IP over everything
 - Then: ARPANET, X.25, DARPA satellite network..
 - Now: ATM, SONET, WDM...

Decentralized Management

- Both a curse and a blessing
 - Important for easy deployment
 - Makes management hard today

Host Attachment

- Clark observes that cost of host attachment may be higher because hosts have to be smart
- But the administrative cost of adding hosts is very low, which is probably more important
 - Plug-and-play kind of behavior....

Cost Effective

- Cheaper than circuit switching at low end
- More expensive than circuit switching at high end
- Not a bad compromise:
 - Cheap where it counts (low-end)
 - More expensive for those who can pay....

Resource Accountability

- Failure!

Internet Motto

We reject kings, presidents, and voting. We believe in rough consensus and running code.”

David Clark

Real Goals

- **Build something that works!**
- Connect existing networks
- Robust in face of failures
- Support multiple types of delivery services
- Accommodate a variety of networks
- Allow distributed management
- Easy host attachment
- Cost effective
- Allow resource accountability

Questions to think about....

- What priorities would a commercial design have?
- What would the resulting design look like?
- What goals are missing from this list?

Modularity

Modularity in Computer Science

**“Modularity based on abstraction
is the way things get done”**

--Barbara Liskov

The Role of Modularity

- We can't build big systems out of spaghetti code
 - Impossible to understand, debug
 - Hard to update
- We need to limit the scope of changes, so that we can update system without rewriting it from scratch
- Modularity is how we limit the scope of changes
 - And understand the system at a higher level

Computer System Modularity

- Partition system into modules
 - Each module has well-defined interface
- Interfaces give flexibility in implementation
 - Changes have limited scope
- Examples:
 - Libraries encapsulating set of functionality
 - Programming language abstracts away CPU
- The trick is to find the *right* modularity
 - The interfaces should be long-lasting
 - If interfaces are changing often, modularity is wrong

Finding the Right Modularity

- Decompose problem into tasks or abstractions
 - Task: *e.g.*, compute a function
 - Abstraction: *e.g.*, provide reliable storage
- Define a module for each task/abstraction
 - Involves defining a clean interface for each module
 - “Clean” means hiding unnecessary details
- Implement system a few times:
 - If interfaces seem to hold, you are on the right track...

Network System Modularity

- The need for modularity still applies
 - **And is even more important!** (why?)
- Network implementations not just distributed across many lines of code
 - Normal modularity “organizes” that code
- Networking is distributed across many machines
 - Hosts
 - Routers

Network Modularity Decisions

- How to break system into modules?
 - Classic decomposition into tasks
- Where are modules implemented?
 - Hosts?
 - Routers?
 - Both?
- Where is state stored?
 - Hosts?
 - Routers?
 - Both?

Leads to three design principles

- How to break system into modules?
 - **Layering**
- Where are modules implemented?
 - **End-to-End Principle**
- Where is state stored?
 - **Fate-Sharing**

Layering

Tasks in Networking

- What does it take to send packets across country?

- Simplistic decomposition:

- Task 1: send along a single wire



- Task 2: stitch these together to go across country



- This gives idea of what I mean by decomposition
 - Next slide presents a much more detailed version

Tasks in Networking (bottom up)

- Electrons on wire
- Bits on wire
- Packets on wire
- Deliver packets across local network
 - Local addresses
- Deliver packets across country
 - Global addresses
- Ensure that packets get there
- Do something with the data

Resulting Modules (layers)

- Electrons on wire (contained in next layer)
- **Bits on wire (Physical)**
- Packets on wire (contained in next layer)
- **Deliver packets across local network (Datalink)**
 - Local addresses
- **Deliver packets across country (Network)**
 - Global addresses
- **Ensure that packets get there (Transport)**
- **Do something with the data (Application)**

Five Layers (top-down)

- **Application:** Providing network support for apps
- **Transport (L4):** (Reliable) end-to-end delivery
- **Network (L3):** Global best-effort delivery
- **Datalink (L2):** Local best-effort delivery
- **Physical:** Bits on wire

- Interactions between these components?
 - Do all components talk to each other?
 - Or are the components limited in their interactions?
- Answer: they are strictly **layered!**

Strictly Layered Dependencies

Applications

...built on...

Reliable (or unreliable) transport

...built on...

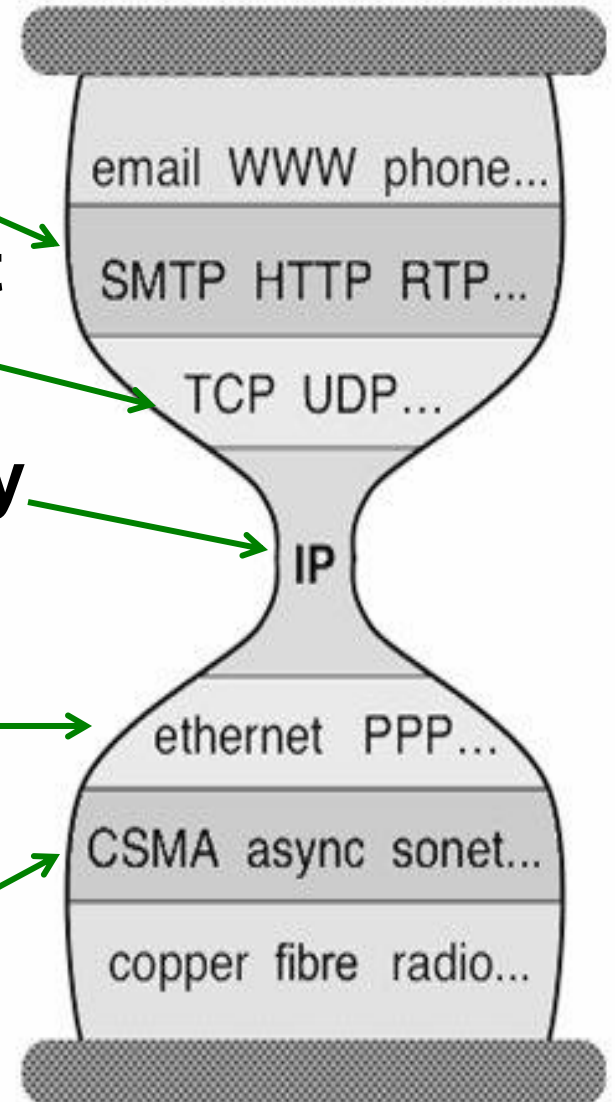
Best-effort global packet delivery

...built on...

Best-effort local packet delivery

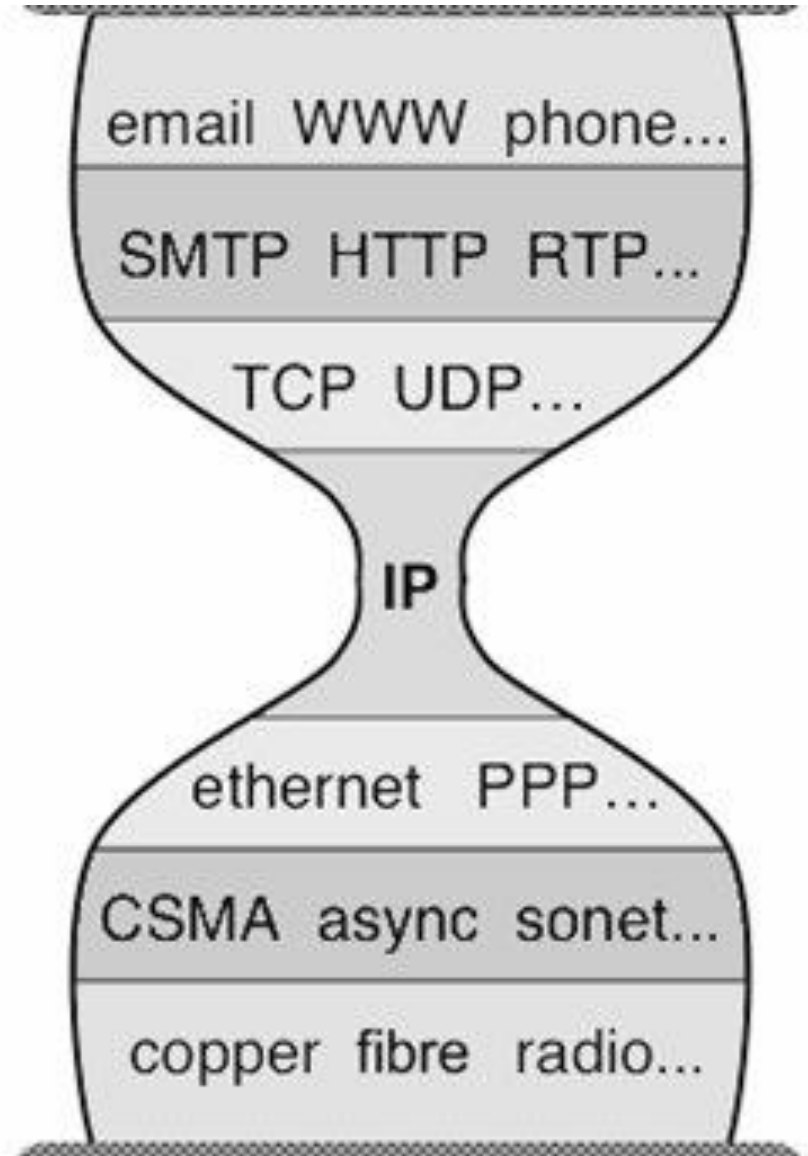
...built on...

Physical transfer of bits



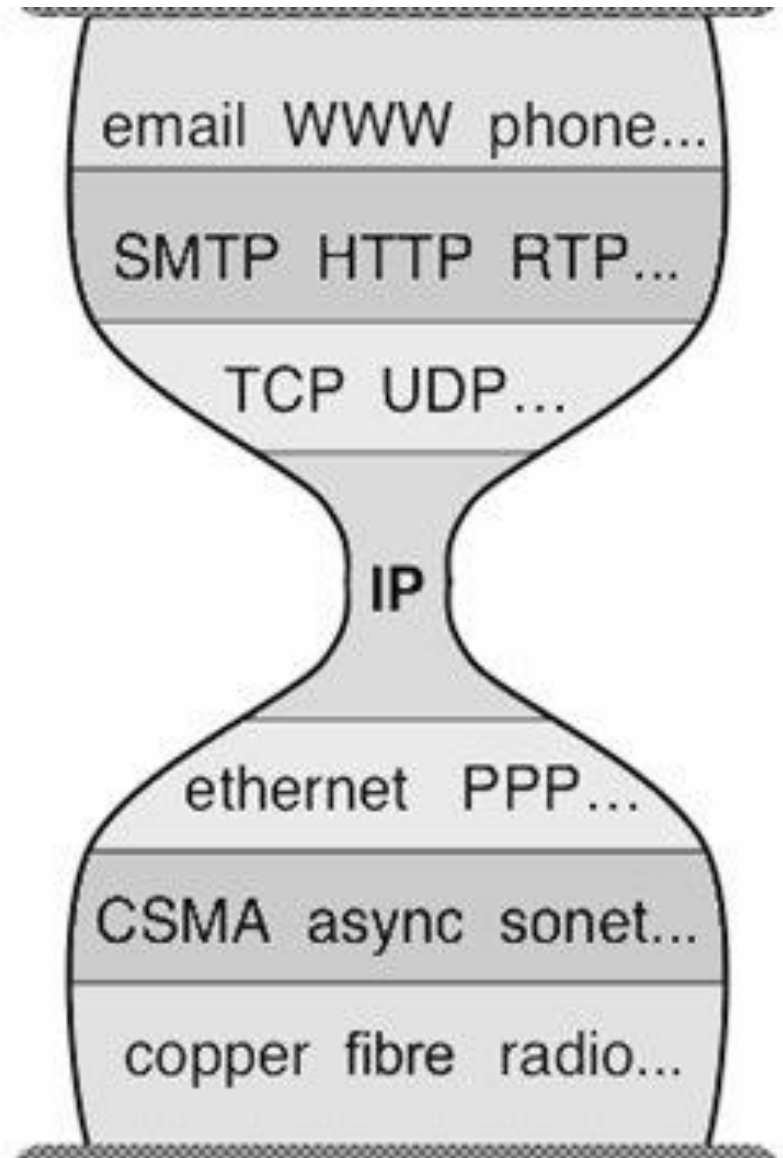
Three Observations

- Each layer:
 - Depends on layer below
 - Supports layer above
 - Independent of others
- Multiple versions in layer
 - Interfaces differ somewhat
 - Components pick which lower-level protocol to use
- But only one IP layer
 - Unifying protocol



Layering Crucial to Internet's Success

- Innovation at most levels
 - Applications (lots)
 - Transport (few)
 - Datalink (few)
 - Physical (lots)
- Innovation proceeded largely in parallel
- Pursued by very different communities
 - Like PL and chip designs



Distributing Layers Across Network

- Layers are simple if only on a single machine
 - Just stack of modules interacting with those above/below
- But we need to implement layers across machines
 - Hosts
 - Routers (switches)
- What gets implemented where?

What Gets Implemented on Host?

- Bits arrive on wire, must make it up to application
- Therefore, all layers must exist at host!

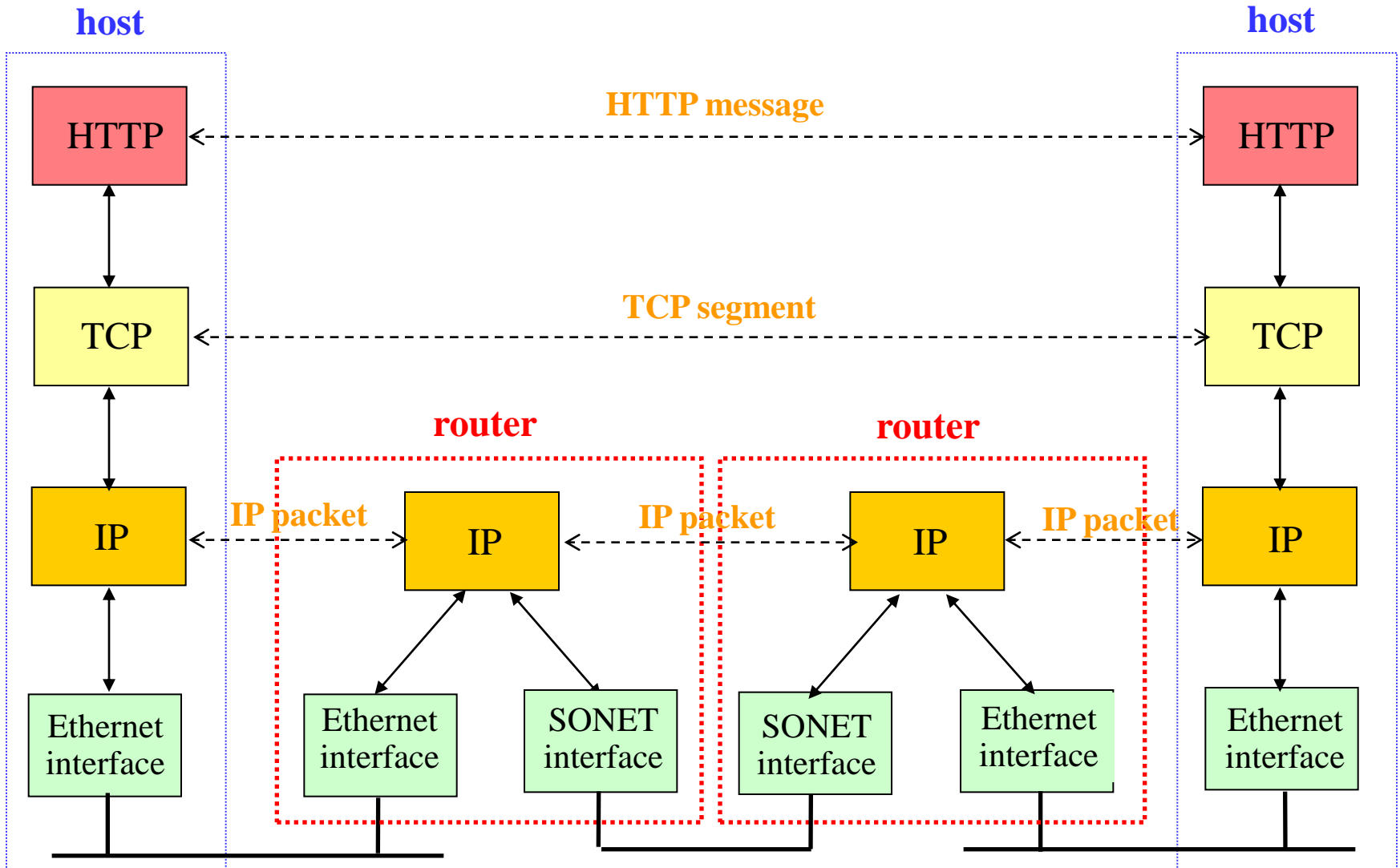
What Gets Implemented on Router?

- Bits arrive on wire
 - Physical layer necessary
- Packets must be delivered to next-hop
 - Datalink layer necessary
- Routers participate in global delivery
 - Network layer necessary
- Routers don't support reliable delivery
 - Transport layer (and above) **not** supported

What Gets Implemented on Switches?

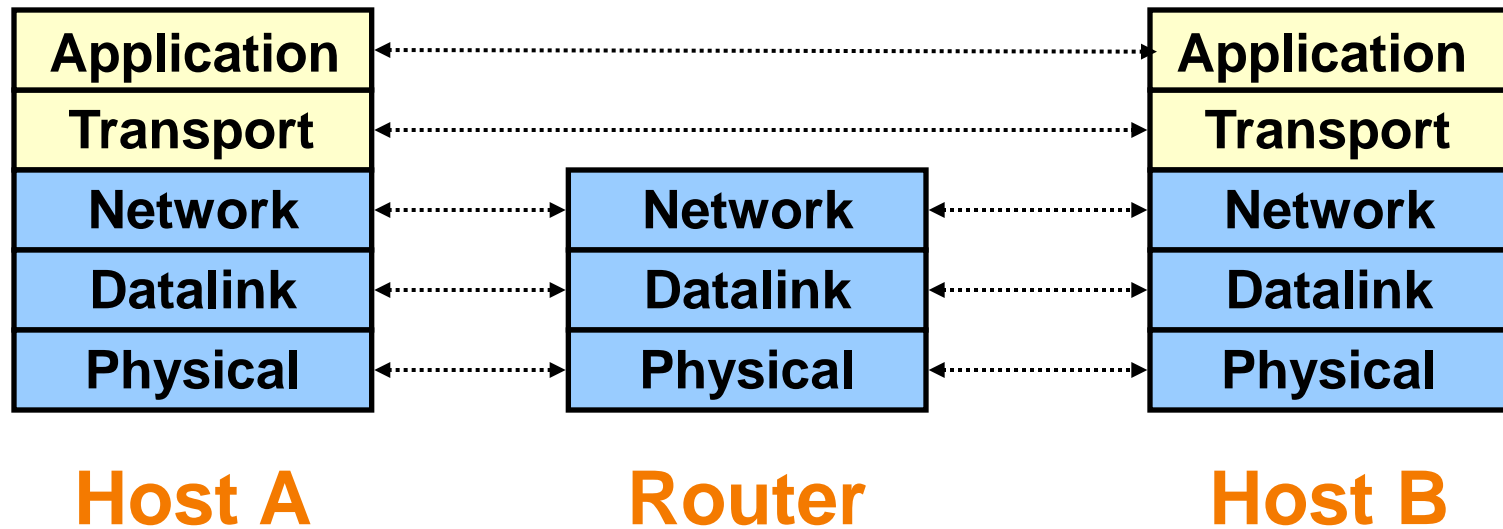
- Switches do what routers do, except they don't participate in global delivery, just local delivery
- They only need to support Physical and Datalink
 - Don't need to support Network layer
- Won't focus on the router/switch distinction
 - When I say switch, I almost always mean router
 - Almost all boxes support network layer these days

Complicated Diagram



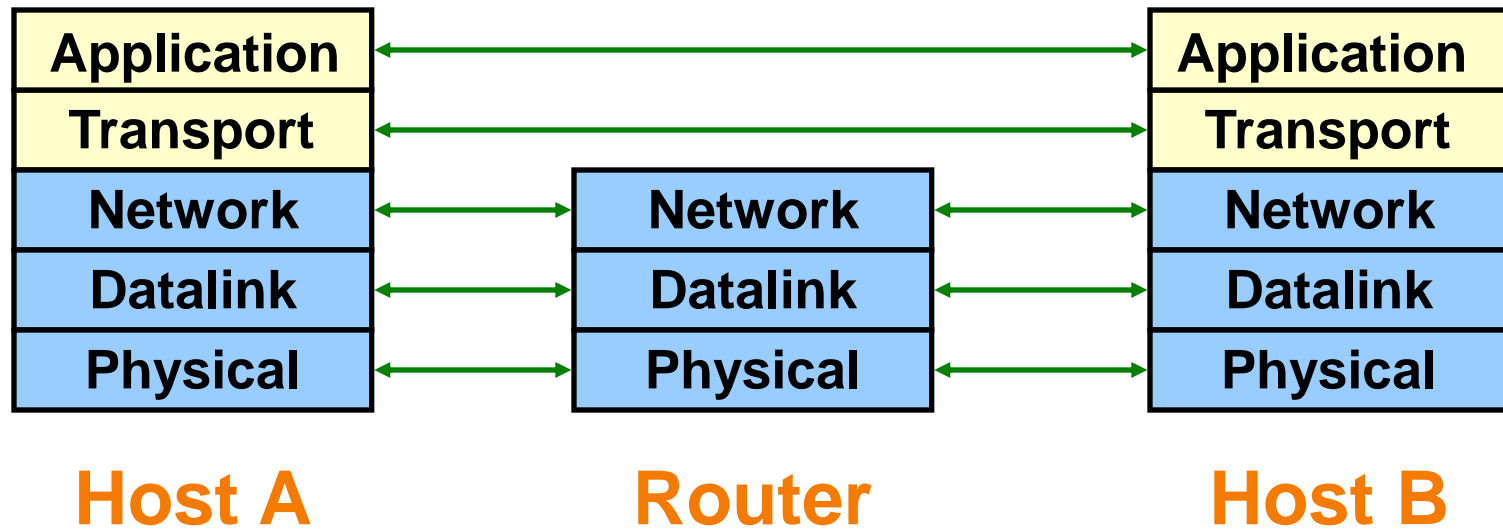
Simple Diagram

- Lower three layers implemented everywhere
- Top two layers implemented only at hosts



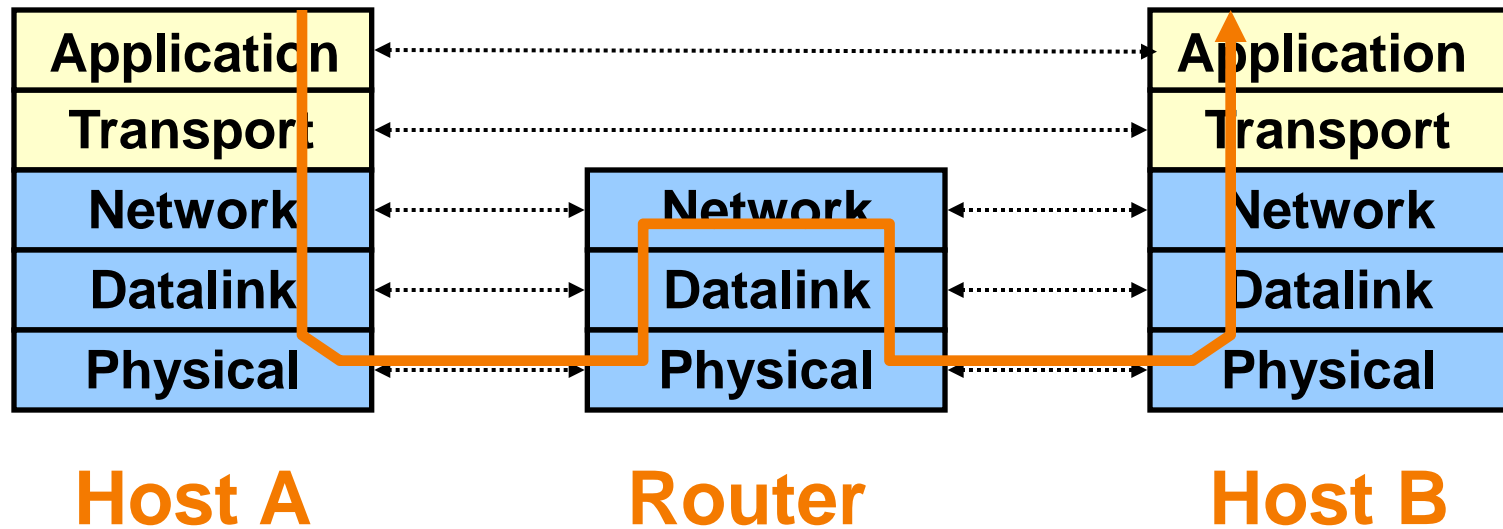
Logical Communication

- Layers interacts with peer's corresponding layer

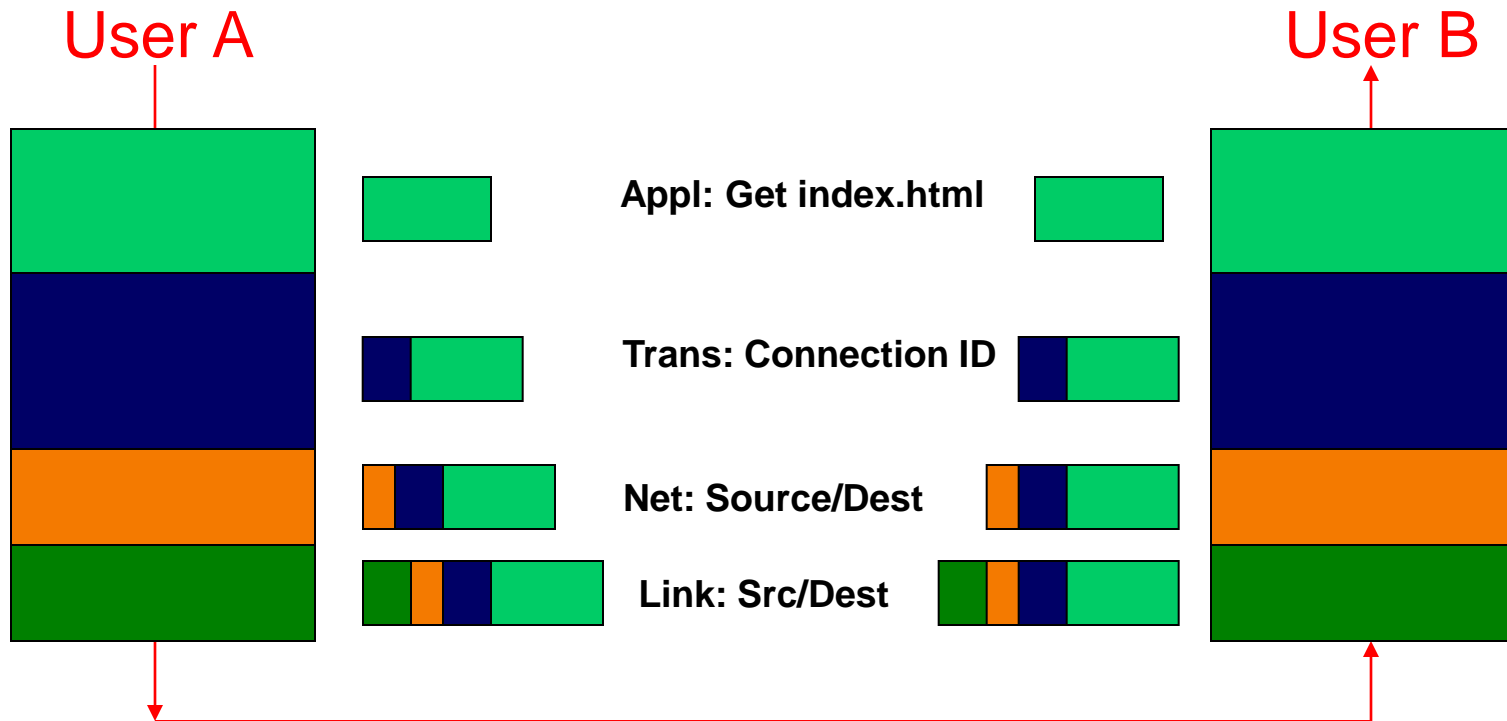
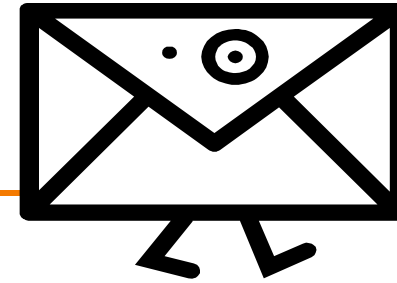


Physical Communication

- Communication goes down to physical network
- Then from network peer to peer
- Then up to relevant layer



Layer Encapsulation



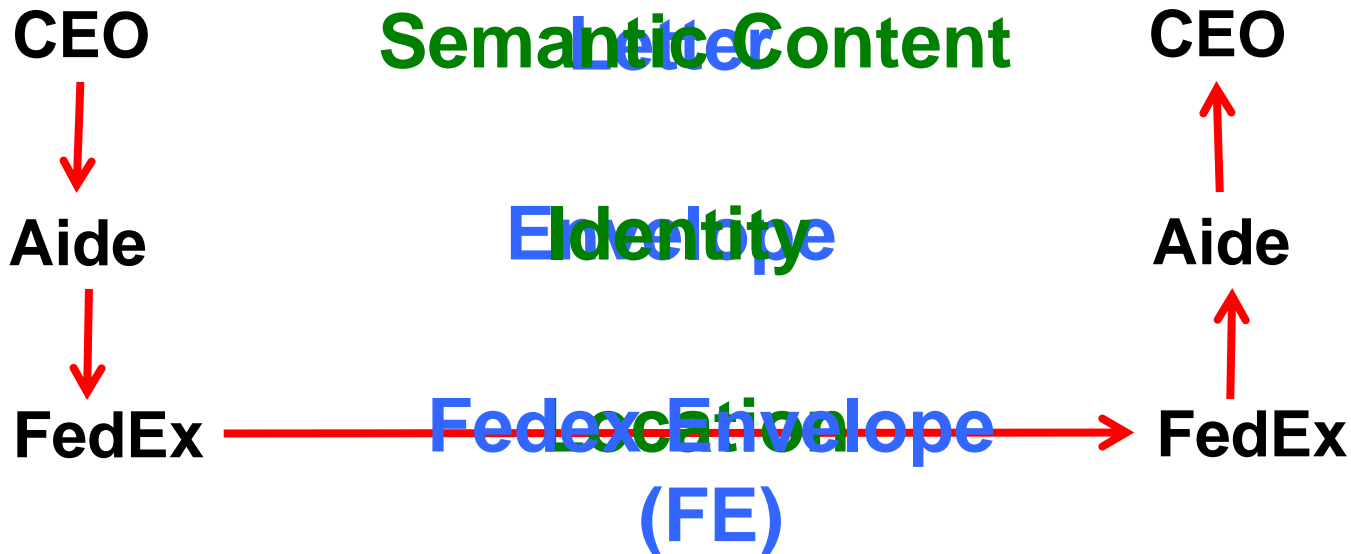
**Common case: 20 bytes TCP header + 20 bytes IP header
+ 14 bytes Ethernet header = 54 bytes overhead**

Example of Layering in the Real World

- CEO A writes letter to CEO B
 - Folds letter and hands it to administrative aide
- Aide:
 - Dear John,**
 - Puts letter in envelope with CEO B's full name
 - Takes to FedEx
 - Your days are numbered.**
- FedEx Office
 - Puts letter in larger envelope
 - Puts name and street address on FedEx envelope
 - Puts package on FedEx delivery truck
- FedEx delivers to other company

The Path of the Letter

“Peers” on each side understand the same things
No one else needs to
Lowest level has most packaging

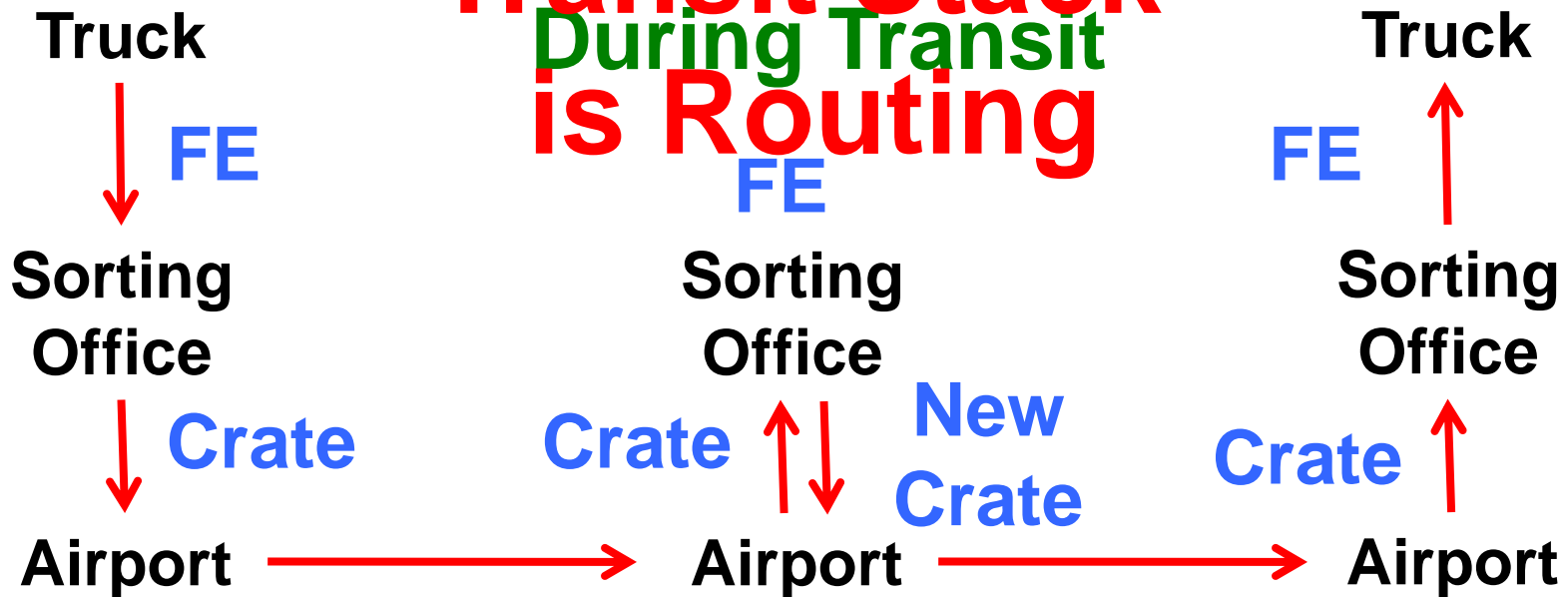


The Path Through FedEx

Higher "Stack"
at Ends

Highest Level of
"Transit Stack"
During Transit
is Routing

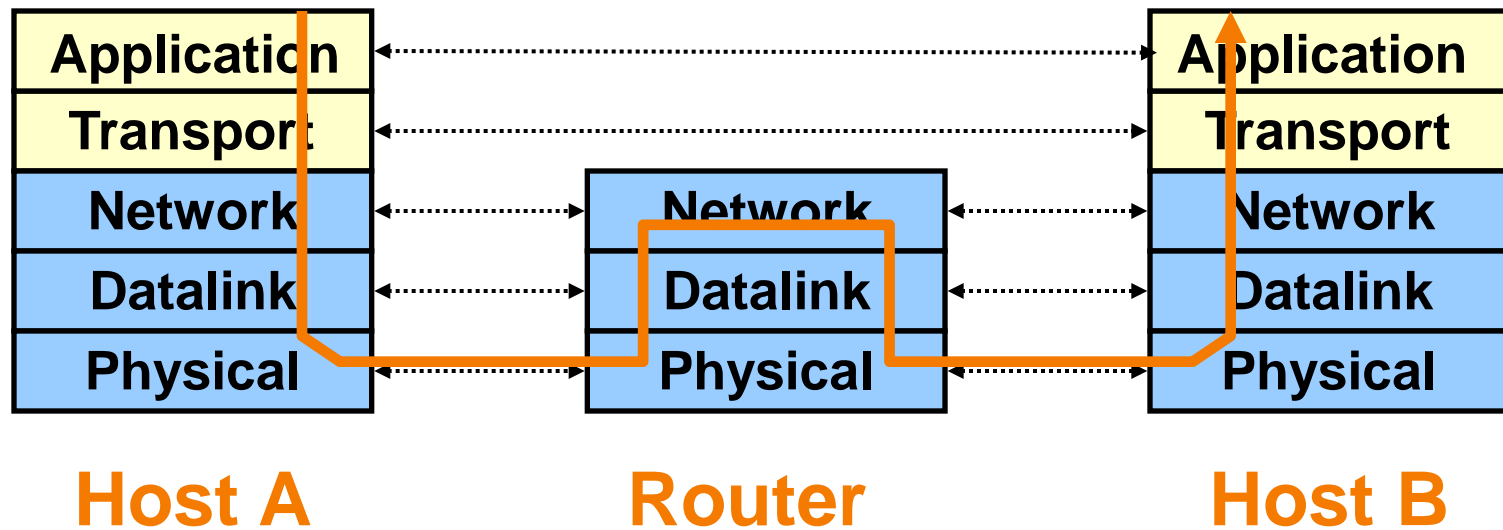
Partial "Stack"



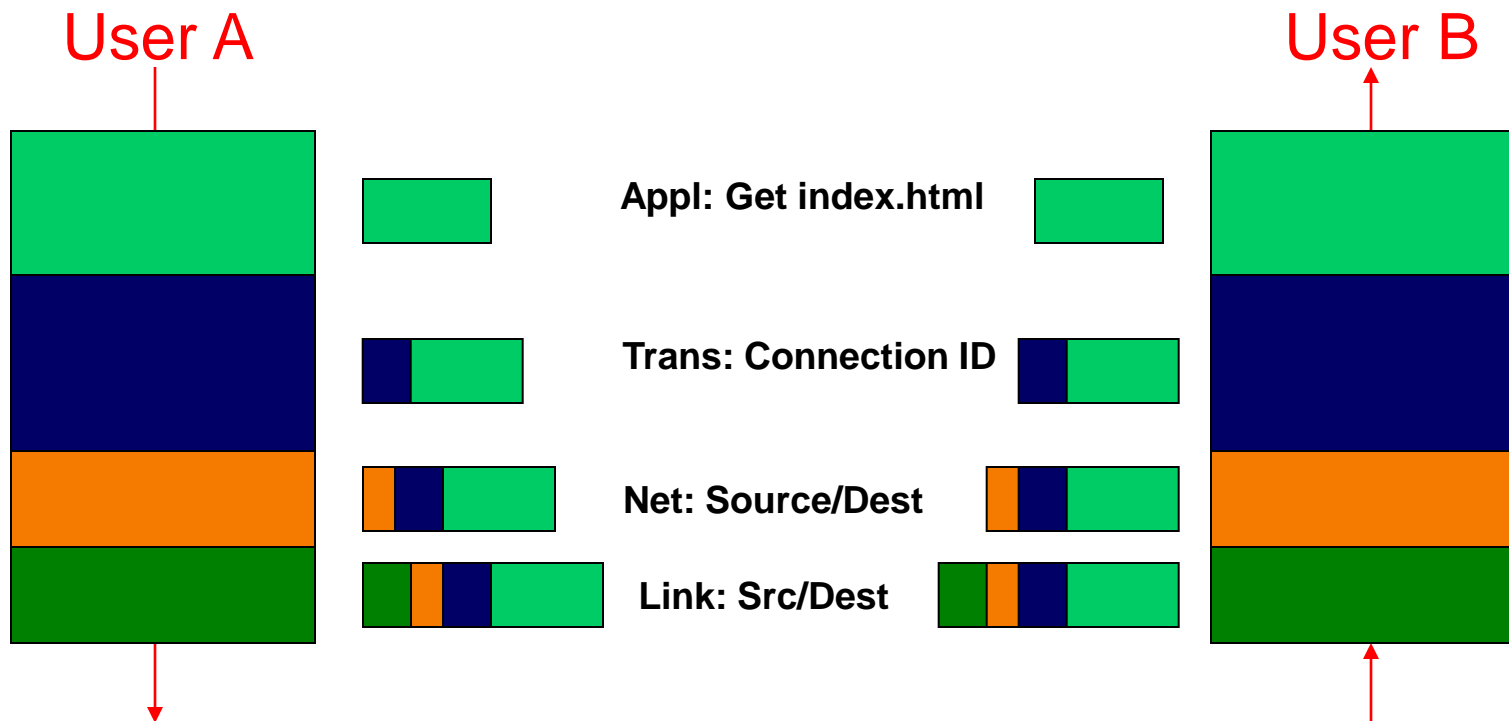
Deepest Packaging (Envelope+FE+Crate)
at the Lowest Level of Transport

Back to Networking Picture

- Communication goes down to physical network
- Then from network peer to peer
- Then up to relevant layer



Back to Encapsulation (Headers)



**Common case: 20 bytes TCP header + 20 bytes IP header
+ 14 bytes Ethernet header = 54 bytes overhead**

Five Minute Break....

Three Internet Design Principles

- How to break system into modules?
 - Layering
- **Where are modules implemented?**
 - **End-to-End Principle**
- Where is state stored?
 - Fate-Sharing

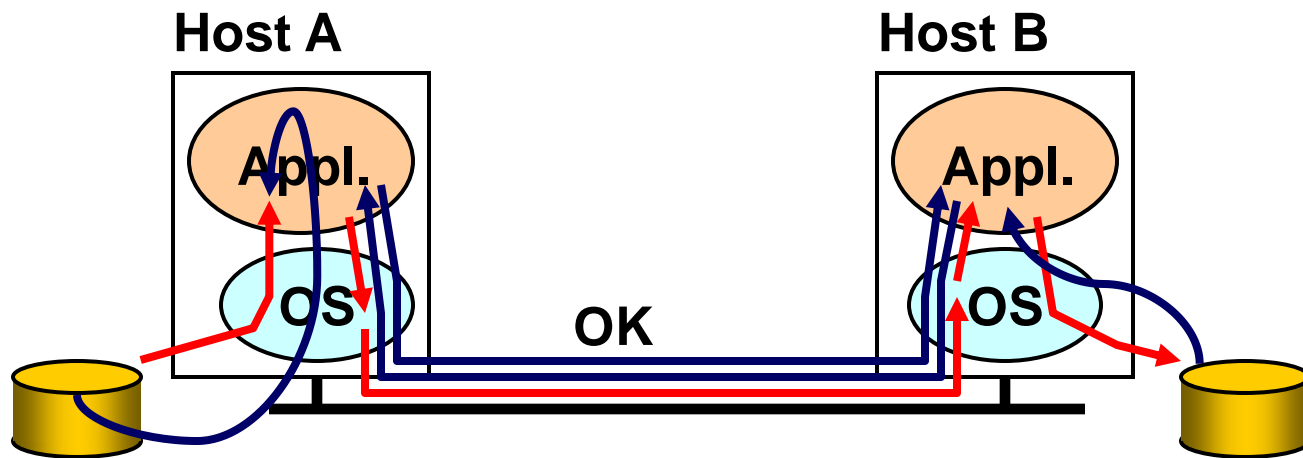
The End-to-End Principle

Everyone believes it, but no one knows
what it means.....

Placing Network Functionality

- Influential paper: “End-to-End Arguments in System Design” by Saltzer, Reed, and Clark (‘84)
 - End-to-end principle
- Basic observation: some types of network functionality can only be correctly implemented ***end-to-end***
- In these cases, end hosts:
 - **Can** satisfy the requirement without network’s help
 - **Must** do so, since can’t ***rely*** on network’s help
- Thus, **don’t** need to implement them in network
 - *Debate about what the network does and doesn’t do...*

Example: Reliable File Transfer



- Solution 1: make each step reliable, and string them together to make reliable end-to-end process
- Solution 2: allow steps to be unreliable, but do end-to-end **check** and try again if necessary

Discussion

- Solution 1 cannot be made perfectly reliable
 - What happens if a network element misbehaves?
 - Receiver has to do the check anyway!
- Solution 2 can also fail, but only if the end system itself fails (i.e., doesn't follow its own protocol)
- Solution 2 only relies on what it can control
 - The endpoint behavior
- Solution 1 requires endpoints trust other elements
 - That's not what reliable means!

Robust (From Clark's Paper)

- As long as the network is not partitioned, two endpoints should be able to communicate
- Failures (excepting network partition) should not interfere with endpoint semantics

Question?

- Should you ever implement reliability in network?
- Perhaps, if needed for reasonable efficiency
 - Don't aim for perfect reliability, but ok to reduce error rate
- If individual links fail 10% of the time, and are traversing 10 links, then E2E error rate is 65%
- Implementing one retransmission on links
 - Link error rate reduced to 1%, E2E error rate is 9.5%

Back to the End-to-End Principle

Implementing such functionality in the network:

- Doesn't reduce host implementation complexity
- Does increase network complexity
- Often imposes delay/overhead on all applications, **even if they don't need functionality**
- However, implementing in network **can** enhance performance in some cases
 - E.g., very lossy link
- Three interpretations of the end-to-end principle

“Only-if-Sufficient” Interpretation

- Don't implement a function at the lower levels of the system unless it can be completely implemented at this level
- *Unless you can relieve the burden from hosts, don't bother*

“Only-if-Necessary” Interpretation

- Don't implement *anything* in the network that can be implemented correctly by the hosts
 - E.g., multicast
- Make network layer absolutely minimal
 - This E2E interpretation trumps performance issues
 - Increases flexibility, since lower layers stay **simple**

“Only-if-Useful” Interpretation

- If hosts can implement functionality correctly, implement it in a lower layer **only** as a performance enhancement
- But do so only if it **does not impose burden** on applications that do not require that functionality

What Does E2E Principle Ignore?

- There are other stakeholders besides users
 - ISP might care about the good operation of their network
 - Various commercial entities
 - Money-chain might require insertion into the network
- The need for middlebox functionality
 - Some functions that, for management reasons, are more easily done in the network.

Three Internet Design Principles

- How to break system into modules?
 - Layering
- Where are modules implemented?
 - End-to-End Principle
- **Where is state stored?**
 - **Fate-Sharing**

Fate-Sharing

Fate-Sharing

- Note that E2E principles relied on “fate-sharing”
 - Invariants break only when endpoints themselves break
 - Minimize dependence on other network elements
- This should dictate placement of storage

General Principle: *Fate-Sharing*

- When storing state in a distributed system, co-locate it with entities that rely on that state
- Only way failure can cause loss of the critical state is if the entity that cares about it also fails ...
 - ... in which case it doesn't matter
- Often argues for keeping *network state* at end hosts rather than inside routers
 - In keeping with End-to-End principle
 - E.g., packet-switching rather than circuit-switching
 - E.g., NFS file handles, HTTP “cookies”

A Cynical View of Distributed Systems

“A distributed system is one in which the failure of a computer you didn't even know existed can render your own computer unusable”

---Leslie Lamport

Decisions and Their Principles

- How to break system into modules
 - **Dictated by Layering**
- Where modules are implemented
 - **Dictated by End-to-End Principle**
- Where state is stored
 - **Dictated by Fate-Sharing**

Question

- If reliability is implemented by the ends, how is it done?
- That's the subject of the next lecture!