Internet Design Principles

EE 122: Intro to Communication Networks

Fall 2006 (MW 4-5:30 in Donner 155)

Vern Paxson

TAs: Dilip Antony Joseph and Sukun Kim

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

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

1

Announcements

- Electronic copy of P&D Chapter 1 available, see announcements Web page
- Homework #1 due date postponed 1 week, now **Wed Sept. 20**. (Also, **bug fixes** for Problem #2.)
- Make sure you're on the mailing list!
- Vern & Dilip away next week @ SIGCOMM, no office hours
  - Sukun lectures Mon Sept. 11
  - **No Lecture Wed Sept. 13**
- Please see note in announcements page re printers to use for hardcopy of the notes

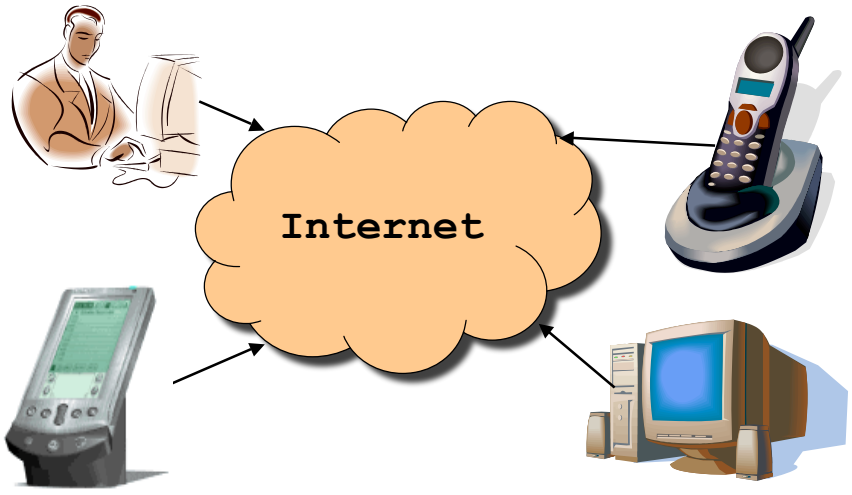
2

Overview

- Roles played by end systems
  - Clients, servers, peer-to-peer
- Architecture & layering
- The End-to-End Principle

3

End System: Computer on the 'Net

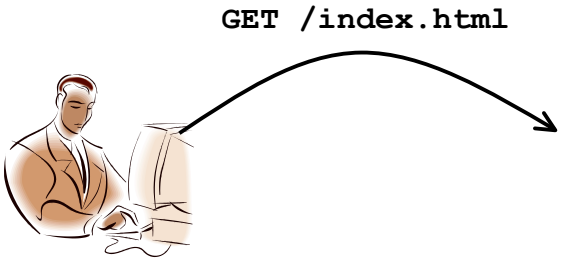


Also known as a "host"…

4

Clients and Servers

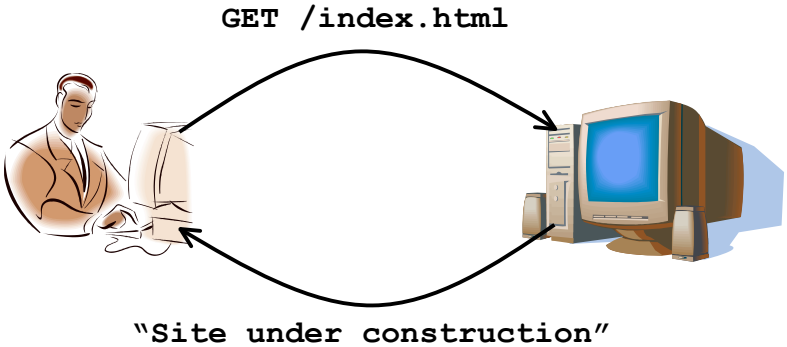
- Client program
  - Running on end host
  - Requests service
  - E.g., Web browser



5

Clients and Servers

- Client program
  - Running on end host
  - Requests service
  - E.g., Web browser
- Server program
  - Running on end host
  - Provides service
  - E.g., Web server



6

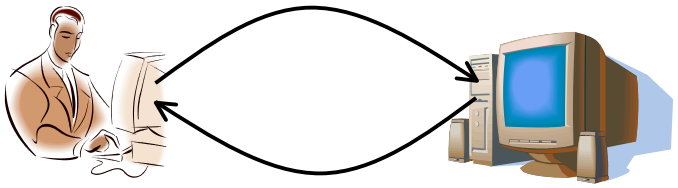
Clients Are Not Necessarily Human

- Example: Web crawler (or spider)
  - Automated client program
  - Tries to discover & download many Web pages
  - Forms the basis of search engines like Google
- Spider client
  - Start with a base list of popular Web sites
  - Download the Web pages
  - Parse the HTML files to extract hypertext links
  - Download these Web pages, too
  - And repeat, and repeat, and repeat...
  - (Per Project #2)

7

Client-Server Communication

- Client "sometimes on"
  - Initiates a request to the server when interested
  - E.g., Web browser on your laptop or cell phone
  - Doesn't communicate directly with other clients
  - Needs to know the server's address
- Server is "always on"
  - Services requests from many client hosts
  - E.g., Web server for the www.cnn.com Web site
  - Doesn't initiate contact with the clients
  - Needs a fixed, well-known address



8

Peer-to-Peer Communication

- No always-on server at the center of it all
  - Hosts can come and go, and change addresses
  - Hosts may have a different address each time
- Example: peer-to-peer file sharing
  - Any host can request files, send files, query to find where a file is located, respond to queries, and forward queries
  - Scalability by harnessing millions of peers
  - Each peer acting as both a client and server

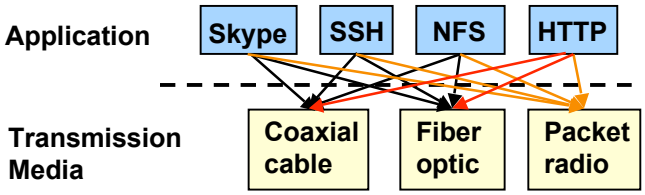
9

Client and Server Processes

- Program vs. process
  - Program: collection of code
  - Process: a running program on a host
- Communication between processes
  - Same end host: inter-process communication
    - Governed by the operating system on the end host
      - (Though can use network protocols for this too)
  - Different end hosts: exchanging messages
    - Governed by the network protocols
- Client and server processes
  - Client process: process that initiates communication
  - Server process: process that waits to be contacted

10

The Problem

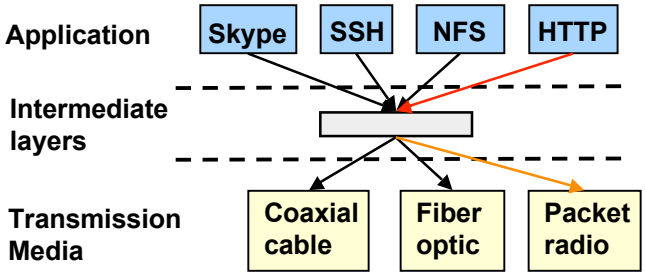


- Re-implement every application for every technology?
- No! But how does the Internet design avoid this?

11

Solution: Intermediate Layers

- Introduce intermediate layers that provide set of abstractions for various network functionality & technologies
  - A new app/media implemented only once
  - Variation on "add another level of indirection"



12

Network Architecture

- Architecture is not the implementation itself
- Architecture is how to organize/structure the elements of the system & their implementation
  - What interfaces are supported
    - Using what sort of abstractions
  - Where functionality is implemented
  - The modular design of the network

13

Computer System Modularity

Partition system into modules & abstractions:

- Well-defined interfaces give flexibility
  - Change implementation of modules
  - Extend functionality of system by adding new modules
- E.g., libraries encapsulating set of functionality
- E.g., programming language + compiler abstracts away not only how the particular CPU works …
  - … but also the basic computational model
- Well-defined interfaces hide information
  - Isolate assumptions
  - Present high-level abstractions
  - **But can impair performance**

14

Network System Modularity

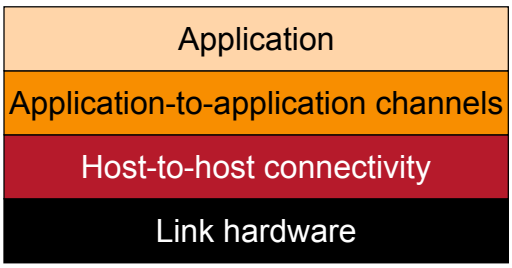
Like software modularity, but:

- Implementation distributed across many machines (routers and hosts)
- Must decide:
  - How to break system into modules
    - **Layering**
  - Where modules are implemented
    - **End-to-End Principle**
- We will address these choices in turn

15

Layering: A Modular Approach

- Paritition the system
  - Each layer solely relies on services from layer below
  - Each layer solely exports services to layer above
- Interface between layers defines interaction
  - Hides implementation details
  - Layers can change without disturbing other layers



16

Properties of Layers (OSI Model)

- **Service**: what a layer does
- **Service interface**: how to access the service
  - Interface for layer above
- **Protocol** (*peer interface*): how peers communicate to achieve the service
  - Set of rules and formats that govern the communication between network elements
  - Does ***not*** govern the implementation on a single machine, but how the layer is implemented ***between*** machines

17

Physical Layer (1)

- **Service**: move signals (information) between two systems connected by a physical link
- **Interface**: specifies how to send bits
- **Protocol**: coding scheme used to represent bits, voltage levels, duration of a bit
- Examples: coaxial cable, optical fiber links; transmitters, receivers

18

(Data)link Layer (2)

- **Service**:
  - Framing (attach frame separators)
  - Deliver data frames from one peer to another
    - Perhaps across *multiple hops*
  - Possible others:
    - arbitrate access to common physical media
    - per-hop reliable transmission
    - per-hop flow control
- **Interface**: send a data unit (packet) to a machine connected to the same physical network
- **Protocols**: addressing (link-layer specific), Medium Access Control (MAC) (e.g., CSMA/CD - *Carrier Sense Multiple Access / Collision Detection*)

19

(Inter)Network Layer (3)

- **Service**:
  - Deliver a packet to specified **inter-network** destination
    - **Inter-network = across multiple networks** / link technologies
  - Perform segmentation/reassembly
    - What if different link technologies have different size limits?
  - Possible others:
    - packet *scheduling*
    - buffer management
- **Interface**: send a packet to a specified internetwork destination
- **Protocols**: define inter-network addresses (globally unique); construct routing tables

20

Transport Layer (4)

- **Service**:
  - Provide end-to-end communication between processes
  - Demultiplexing of communication between hosts
  - Possible others:
    - Reliability in the presence of errors
    - Timing properties
    - Rate adaption (flow-control, congestion control)
- **Interface**: send message to specific destination
- **Protocol**: implements reliability and flow control
- Examples: TCP and UDP

21

Application Layer (7 - not 5!)

- **Service**: any service provided to the end user
- **Interface**: depends on the application
- **Protocol**: depends on the application
- Examples: Skype, SMTP (email), HTTP (Web), Counter-Strike
- What happened to layers 5 & 6?
  - "Session" and "Presentation" layers
  - Formalized (OSI/ISO; see text), but not widely used

22

5 Minute Break

Questions Before We Proceed?

23

Drawbacks of Layering

- Layer N may duplicate lower level functionality
  - E.g., error recovery to retransmit lost data
- Layers may need same information
  - E.g., timestamps, maximum transmission unit size
- Strict adherence to layering may hurt performance
  - E.g., hiding details about what is really going on
- Some layers are not always cleanly separated
  - Inter-layer dependencies for performance reasons
  - Some dependencies in standards (header checksums)
- Headers start to get really big
  - Sometimes header bytes >> actual content

24

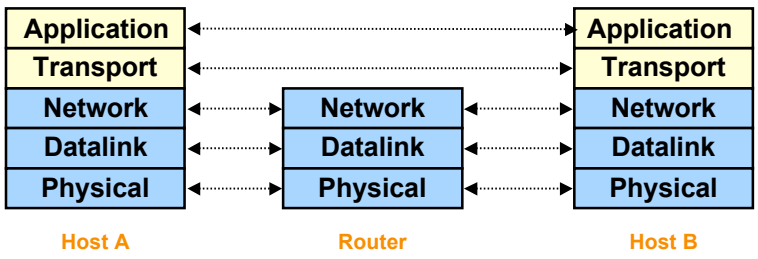
Layer Violations

- Sometimes the gains from not respecting layer boundaries too great to resist
- Can occur with higher-layer entity inspecting lower-layer information:
  - E.g., TCP-over-wireless system that monitors wireless link-layer information to try to determine whether packet loss due to congestion or corruption
- Can occur with lower-layer entity inspecting higher-layer information
  - E.g., firewalls, NATs (network address translators), "transparent proxies"
- Just as with in-line assembly code, can be **messy** and paint yourself into a corner (you know too much)

25

Who Does What?

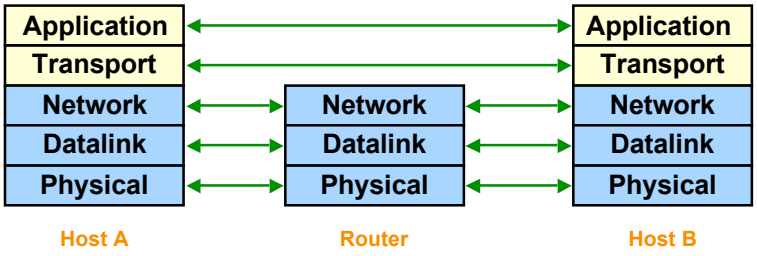
- Five layers
  - Lower three layers implemented everywhere
  - Top two layers implemented only at hosts



26

Logical Communication

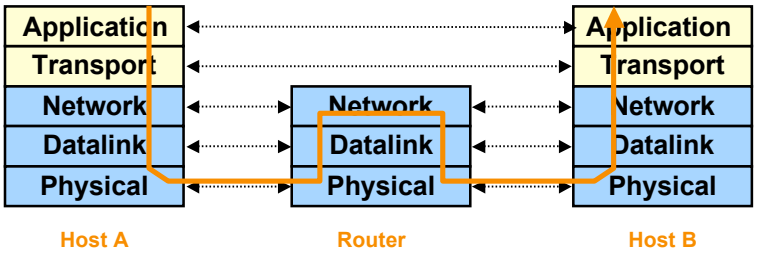
- Layers interacts with peer's corresponding layer



27

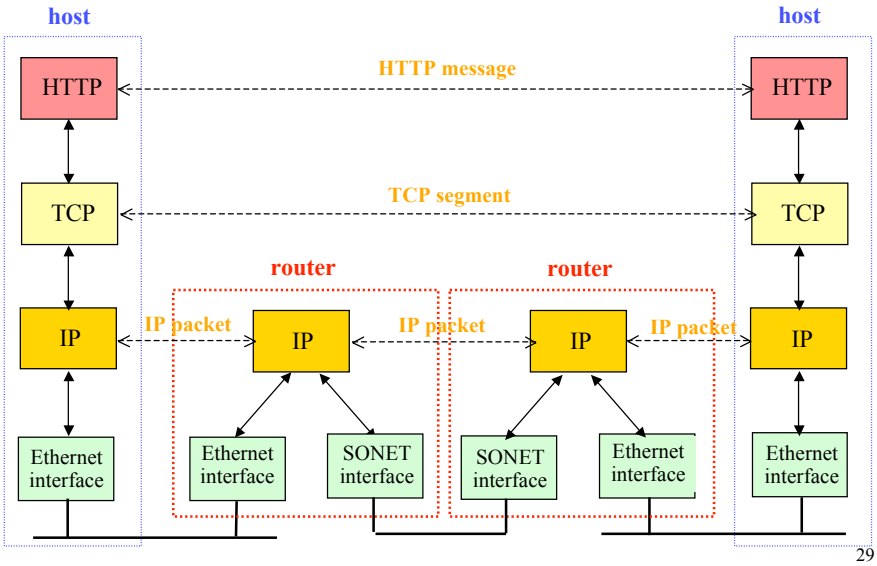
Physical Communication

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



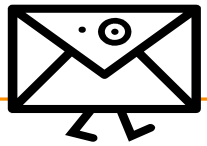
28

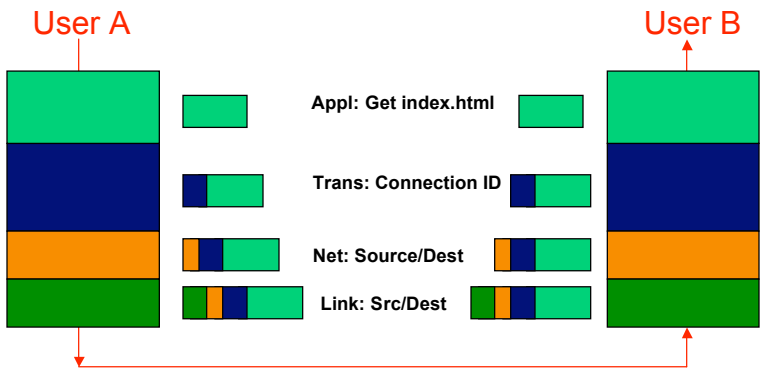
IP Suite: End Hosts vs. Routers



29

Layer Encapsulation

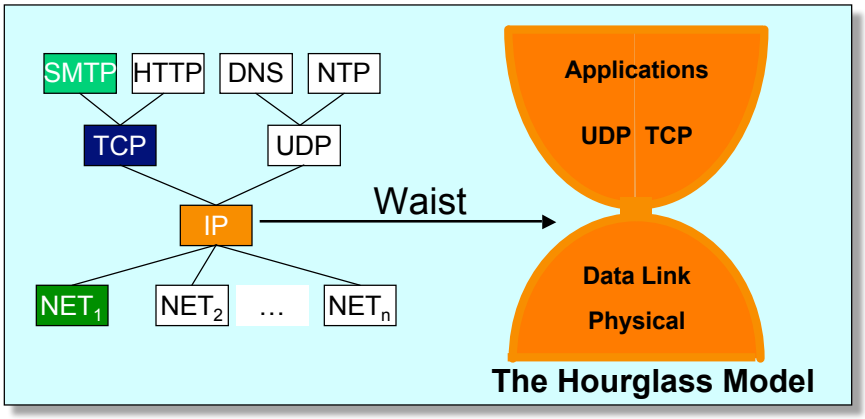




Common case: 20 bytes TCP header + 20 bytes IP header + 14 bytes Ethernet header = 54 bytes overhead

30

The Internet *Hourglass*



There is just one network-layer protocol, **IP**.
The "narrow waist" facilitates interoperability.

31

Implications of Hourglass

Single Internet-layer module (**IP**):

- Allows arbitrary networks to interoperate
  - Any network technology that supports IP can exchange packets
- Allows applications to function on all networks
  - Applications that can run on IP can use any network
- Supports simultaneous innovations above and below IP
  - But changing IP itself, i.e., **IPv6**, very involved

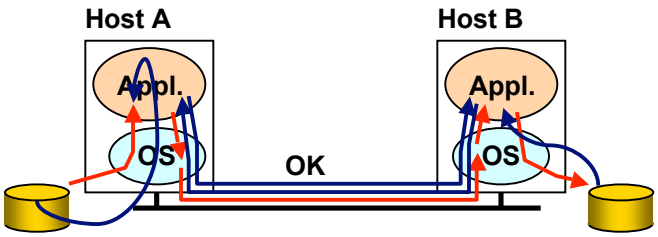
32

Placing Network Functionality

- Hugely influential paper: "End-to-End Arguments in System Design" by Saltzer, Reed, and Clark ('84)
- Basic observation: some types of network functionality can only be correctly implemented end-to-end
- Because of this, end hosts:
  - Can satisfy the requirement without network's help
  - Will/**must** do so, since can't depend on network's help
- Therefore **don't** go out of your way to implement them in the network

33

Example: Reliable File Transfer



- Solution 1: make each step reliable, and then concatenate them
- Solution 2: end-to-end **check** and retry

34

Discussion

- Solution 1 is incomplete
  - What happens if any network element misbehaves?
  - Receiver has to do the check anyway!
- Solution 2 is complete
  - Full functionality can be entirely implemented at application layer with no need for reliability from lower layers
- *Is there any need to implement reliability at lower layers?*

35

Summary of End-to-End Principle

Implementing this functionality in the network:

- Doesn't reduce host implementation complexity
- Does increase network complexity
- Probably imposes delay and 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

36

Conservative Interpretation of E2E

- "Don't implement a function at the lower levels of the system unless it can be completely implemented at this level" (Peterson and Davie)
- Unless you can relieve the burden from hosts, then don't bother

37

Radical Interpretation of E2E

- Don't implement anything in the network that can be implemented correctly by the hosts
  - E.g., multicast
- Make network layer absolutely minimal
  - E2E principle trumps performance issues

38

Moderate Interpretation

- Think twice before implementing functionality in the network
- If hosts can implement functionality correctly, implement it 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

39

Related Notion of *Fate-Sharing*

- Idea: when storing state in a distributed system, keep it co-located with the entities that ultimately rely on the state
- Fate-sharing is a technique for dealing with failure
  - Only way that 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 (packet-switched rather than circuit-switched)
  - In keeping with End-to-End principle

40

Summary

- Layered architecture as a powerful means for organizing complex networks
  - Though layering has its drawbacks too
- Unified Internet layering (Application/Transport/ Internetwork/Link/Physical) decouples apps from networks
- E2E argument encourages us to keep IP simple
- Commercial realities (need to control the network) can greatly stress this

41

Next Lecture

- How applications **use** the network: **sockets**
  - Lecture by Sukun: Vern & Dilip away at SIGCOMM
- Read *Stevens*: 3.1-3.7, 4.1-4.6, 6.1-6.3
- **No lecture on Wed Sept. 13**
- No office hours for Vern & Dilip next week (but use email)

42