CS162
Operating Systems and Systems Programming
Lecture 14

Protocols, Layering and e2e Argument

March 14, 2011
Ion Stoica
http://inst.eecs.berkeley.edu/~cs162

Goals for Today

- Finish Page Replacement
- Working Set/Thrashing
- Introduction to networking

Note: Some slides and/or pictures in the following are adapted from slides ©2005 Silberschatz, Galvin, and Gagne. Many slides generated from my lecture notes by Vern Paxson, and Scott Shenker.

What Is A Protocol?

- A protocol is an agreement on how to communicate
- Includes
  - Syntax: how a communication is specified & structured
    - » Format, order messages are sent and received
  - Semantics: what a communication means
    - » Actions taken when transmitting, receiving, or when a timer expires

Examples of Protocols in Human Interactions

- Telephone
  1. (Pick up / open up the phone.)
  2. Listen for a dial tone / see that you have service.
  3. Dial.
  4. Should hear ringing …
  5. Callee: "Hello?"
  6. Caller: "Hi, it's Alice …."
     Or: "Hi, it's me" (← what's *that* about?)
  7. Caller: "Hey, do you think … blah blah blah …" **pause**
  8. Callee: "Yeah, blah blah blah …" **pause**
  9. Caller: Bye
  10. Callee: Bye
  11. Hang up

Examples of Protocols in Human Interactions

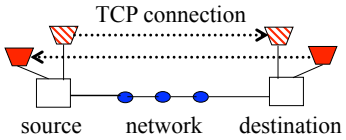
- Asking a question
  1. Raise your hand.
  2. Wait to be called on.
  3. Or: wait for speaker to **pause** and vocalize

The Internet Protocol (IP): "Best-Effort" Packet Delivery

- Datagram packet switching
  - Send data in packets
  - Header with source & destination address
- Service it provides:
  - Packets may be lost
  - Packets may be corrupted
  - Packets may be delivered out of order

Example: Transmission Control Protocol (TCP)

- Communication service
  - Ordered, reliable byte stream
  - Simultaneous transmission in both directions
- Key mechanisms at end hosts
  - Retransmit lost and corrupted packets
  - Discard duplicate packets and put packets in order
  - Flow control to avoid overloading the receiver buffer
  - Congestion control to adapt sending rate to network load



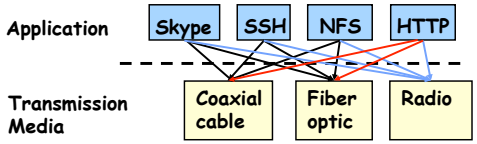
Protocol Standardization

- Ensure communicating hosts speak the same protocol
  - Standardization to enable multiple implementations
  - Or, the same folks have to write all the software
- Standardization: Internet Engineering Task Force
  - Based on working groups that focus on specific issues
  - Produces "Request For Comments" (RFCs)
    - » Promoted to standards via rough consensus and running code
  - IETF Web site is ***http://www.ietf.org***
  - RFCs archived at ***http://www.rfc-editor.org***
- De facto standards: same folks writing the code
  - P2P file sharing, Skype, <your protocol here>…

Layering: The Problem

- Many different applications
  - email, web, P2P, etc.
- Many different network styles and technologies
  - Circuit-switched vs packet-switched, etc.
  - Wireless vs. wired vs optical, etc.
- How do we organize this mess?

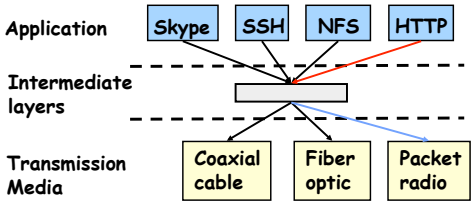
The Problem (cont'd)



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

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"



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

Software System Modularity

Partition system into modules & abstractions:

- Well-defined interfaces give flexibility
  - ***Hides*** implementation - thus, it can be freely changed
  - 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**

Network System Modularity

Like software modularity, but:

- Implementation distributed across many machines (routers and hosts)
- Must decide:
  - How to break system into modules
    - » **Layering**
  - What functionality does each module implement
    - » **End-to-End Principle**
  - Where state is stored
    - » **Fate-sharing**
- We will address these choices in turn

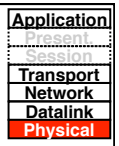
Layering: A Modular Approach

- Partition 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

Properties of Layers (OSI Model)

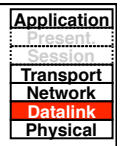
- **Service**: what a layer does
- **Service interface**: how to access the service
  - Interface for layer above
- **Protocol** (*peer interface*): how peers communicate to achieve the service
  - Set of rules and formats that specify the communication between network elements
  - Does ***not*** specify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

Physical Layer (1)



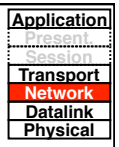
- **Service**: move information between two systems connected by a physical link
- **Interface**: specifies how to send and receive bits
- **Protocol**: coding scheme used to represent a bit, voltage levels, duration of a bit
- Examples: coaxial cable, optical fiber links; transmitters, receivers

(Data) Link Layer (2)



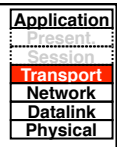
- **Service**:
  - Enable end hosts to exchange atomic messages with one another
    - » Using abstract addresses (i.e., **not** just direct physical connections)
  - Perhaps over *multiple physical links*
    - » But using the same *framing* (headers/trailers)
  - Possible other services:
    - » arbitrate access to common physical media
    - » reliable transmission, flow control
- **Interface**: send messages (*frames*) to other end *hosts*; receive messages addressed to end host
- **Protocols**: addressing, routing, Media Access Control (MAC) (e.g., CSMA/CD - *Carrier Sense Multiple Access / Collision Detection*)

(Inter) Network Layer (3)



- **Service**:
  - Deliver packets to specified **inter-network** destination
    - » **Inter-network** = across **multiple layer-2 networks**
  - Works *across networking technologies* (e.g., Ethernet + 802.11 + Frame Relay …)
    - » No longer the same framing all the way
  - Possible other services:
    - » packet *scheduling/priority*
    - » buffer management
- **Interface**: send *packets* to specified inter-network destinations; receive packets destined for end host
- **Protocols**: define inter-network addresses (globally unique); construct routing tables

Transport Layer (4)



- **Service**:
  - Provide end-to-end communication between processes
  - Demultiplexing of communication between hosts
  - Possible other services:
    - » Reliability in the presence of errors
    - » Timing properties
    - » Rate adaption (flow-control, congestion control)
- **Interface**: send message to specific process at given destination; local process receives messages sent to it
- **Protocol**: port numbers, perhaps implement reliability, flow control, packetization of large messages, framing
- Examples: TCP and UDP

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), Halo, BitTorrent …

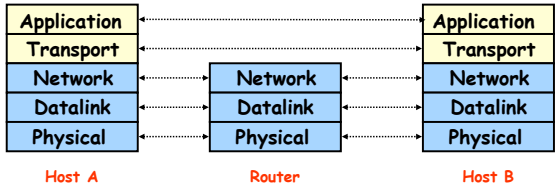
- What happened to layers 5 & 6?
  - "Session" and "Presentation" layers
  - Part of ***OSI*** architecture, but not Internet architecture

5 Minute Break

Questions Before We Proceed?

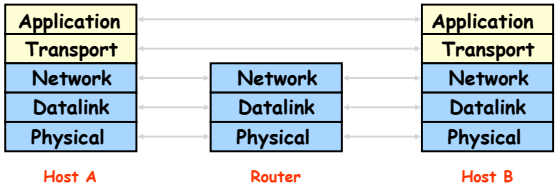
Who Does What?

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



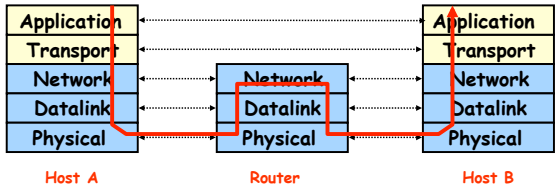
Logical Communication

- Layers interacts with peer's corresponding layer

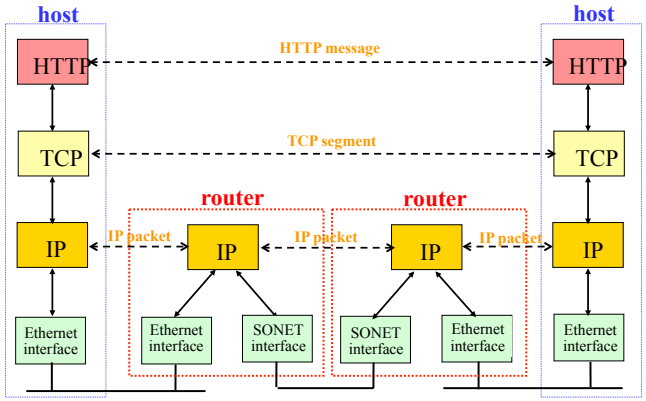


Physical Communication

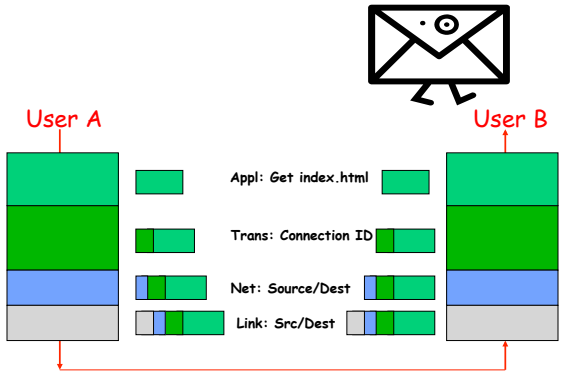
- Communication goes down to physical network
- Then from network peer to peer
- Then up to relevant layer



IP Suite: End Hosts vs. Routers

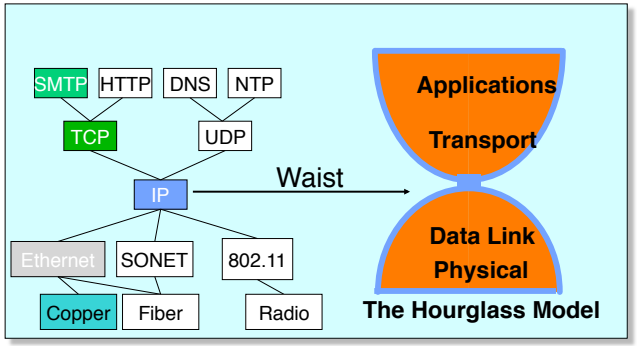


Layer Encapsulation



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

The Internet *Hourglass*



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

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

Drawbacks of Layering

- Layer N may duplicate layer N-1 functionality
  - E.g., error recovery to retransmit lost data
- Layers may need same information
  - E.g., timestamps, maximum transmission unit size
- Layering can 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

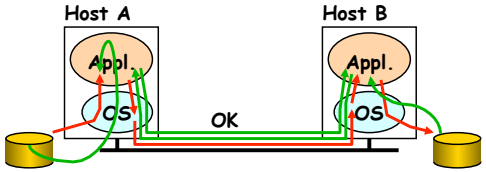
Placing Network Functionality

- Hugely influential paper: "End-to-End Arguments in System Design" by Saltzer, Reed, and Clark ('84)
- "Sacred Text" of the Internet
  - Endless disputes about what it means
  - Everyone cites it as supporting their position

Basic Observation

- Some types of network functionality can only be correctly implemented end-to-end
  - Reliability, security, etc
- Because of this, end hosts:
  - Can satisfy the requirement without network's help
  - Will/**must** do so, since can't ***rely*** on network's help
- Therefore **don't** go out of your way to implement them in the network

Example: Reliable File Transfer



- Solution 1: make each step reliable, and then concatenate them
- Solution 2: end-to-end **check** and try again if necessary

Discussion

- Solution 1 is incomplete
  - What happens if memory is corrupted?
  - 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?*
  - Well, it could be more efficient

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 losy link

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
- Unless you can relieve the burden from hosts, don't bother

Moderate Interpretation

- Think twice before implementing functionality in the network
- 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

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
  - In keeping with End-to-End principle
  - E.g., packet-switching rather than circuit-switching
  - E.g., NFS file handles, HTTP "cookies"

Summary

- Roles of
  - Standardization
  - Clients, servers, peer-to-peer
- 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 networking technologies
- E2E argument encourages us to keep IP simple
  - Commercial realities (need to control the network) can greatly stress this