CS 3204
Operating Systems

Lecture 27

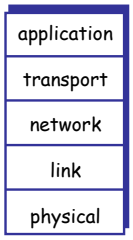
Godmar Back

Announcements

- Project 4 due Dec 6, 11:59pm
- Reading assignments:
  - Chapters 14 & 15
- Sample Final Exam posted
- Next Tuesday:
  - Q&A Session for Final + Teaching Evaluations

Layered Protocol Architecture

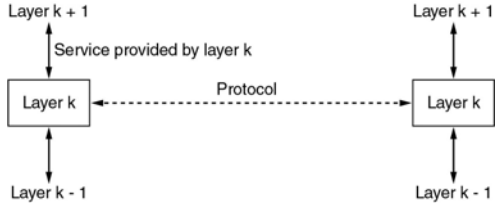
- What is a layered architecture?
- Motivation
- Terminology
- Reference Models
- Implementation Issues



Advantages of Layering

- Decomposition
  - Masters complexity
- Encapsulation
  - Hiding of implementation details
- Evolution
  - Layers can change/be replaced
  - Alternative implementations can be added, possibly coexist
- Robustness
  - Testing layers independently increases confidence

Services vs Protocols



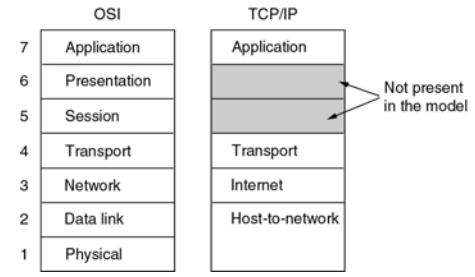
(horizontal component)

- Layer *k* may interact with peer layer *k* only via protocols

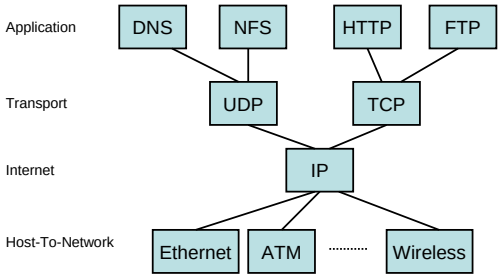
Interfaces vs Protocols

- Duality
  - Both describe rules regarding order and format of "communication" between entities
- Difference:
  - Interface – vertical: between layers
  - Protocol – horizontal: between peers
- NB: Term "protocol" is sometimes used to describe module implementing a layer

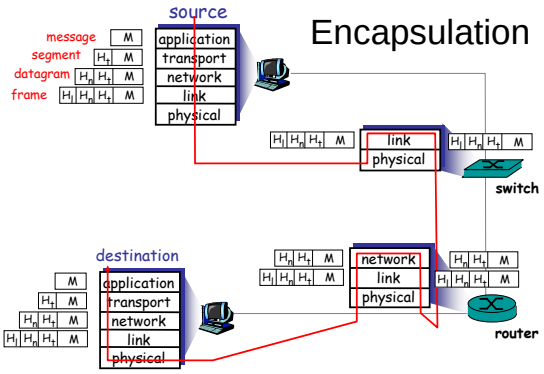
TCP/IP Reference Model



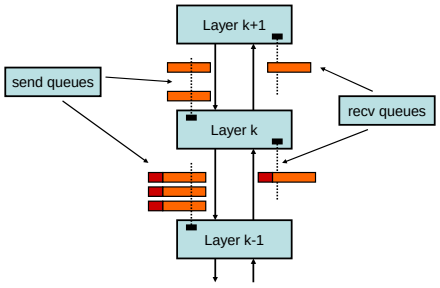
TCP/IP Hourglass View



Encapsulation



Queuing

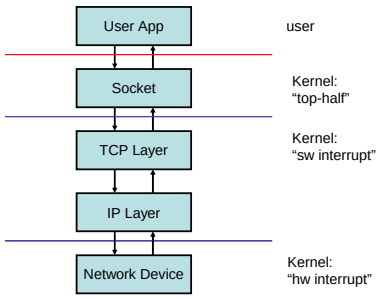


General Implementation Issues

- Who schedules a layer's processing:
  - On send: application thread
    - What if queues are full? Blocking vs. nonblocking
    - Who does retransmit if necessary?
  - On receive: interrupt-driven
    - Not all processing done right away: some delayed processing
    - Sometimes interrupts can be too slow; polling based approach used instead
- Memory management/Protection Issues:
  - How to you prepend headers?
  - How do you strip headers?
  - When are copies needed?
  - Who allocates and frees packet buffers?

Typical Implementation

may cross multiple boundaries!



Demultiplexing

- End systems must decide which layer instances should process an incoming packet.
- Layer k has "type information" in header to say which instance of layer k+1 to pick.
- Issues: speed and flexibility

Ethernet Frame as received

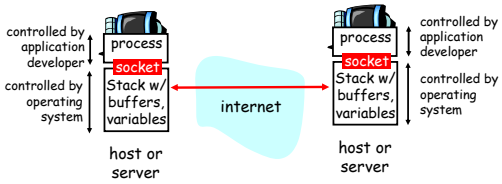
Ethernet Type	IP Protocol	TCP Port	Http Request
.....0800 ->IP	.....06 ->TCP.....	.....50 ->http.....	.....GET /~gback.....
Ethernet Header	IP Header	TCP Header	HTTP Request

Socket Programming

UDP & TCP

Network Socket Programming

Socket: (narrow definition:) a door between application process and end-end-transport protocol (UDP or TCP)



Socket Programming

Socket API

- introduced in BSD 4.1 UNIX, 1981
- explicitly created, used, released by apps
- used for both local and remote communication

socket

a *host-local*, *application-created*, *OS-controlled* interface (a "door") into which application process can both send and receive messages to/from another application process

BSD Socket API

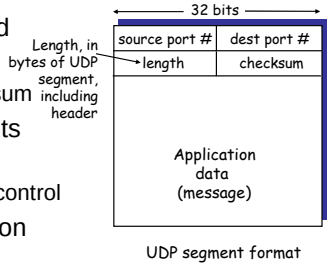
- API – Application Programming Interface
  - Provides access to services
  - Specified in C language
  - Implemented on many platforms in one way or the other
    - (Windows: WinSock2, CSocket MFC classes for BSD-like look)
- Sockets (in Unix) are file descriptors
  - General idea: writing to the socket is sending data to network, reading from socket is receiving data
  - Good because read(2), write(2), close(2) and others (select(2), poll(2), ioctl(2), SIGIO, fcntl(2)) can be reused
  - Bad because suggest orthogonality if where there is none
- Other languages provide separate mapping, often thin veneers over BSD sockets (e.g., java.net.Socket)

Addressing

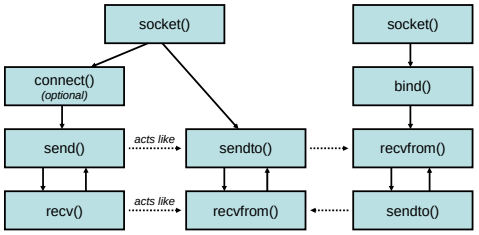
- For UDP/IP or TCP/IP socket communication, generally need 4 parameters:
  - Source Identifier (32-bit IP Address)
  - Source Port (16-bit)
  - Destination Identifier (32-bit IP Address)
  - Destination Port (16-bit)
- Notice that the relationship of "local" and "remote" (also called "peer") to source/destination depends on direction of communication
- Note:
  - UDP uses only Destination (IP+Port) for demultiplexing
  - TCP uses Source + Destination
    - (quadruple: Src IP, Src Port, Dst IP, Dest Port)

UDP

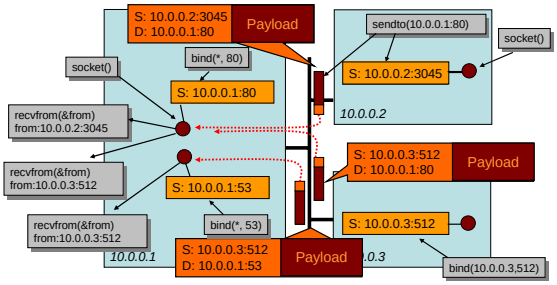
- Service provided
  - Demultiplexing
  - Payload checksum
- Passes segments straight to IP
  - No congestion control
- So simple it fits on one slide



UDP Sockets: Overview



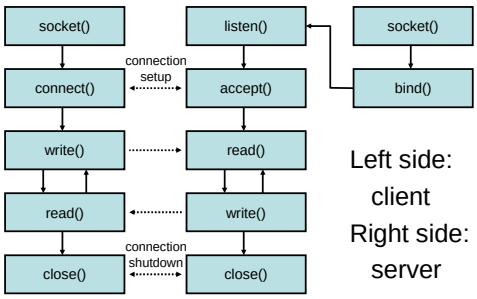
UDP Demultiplexing



TCP Sockets

- Provide reliable byte-stream abstraction
  - In-order, reliable delivery of *bytes*
- Connection-oriented
  - Client must connect(2)
  - Server performs “passive open” using accept(2)

TCP Sockets: Overview



Left side: client

Right side: server

TCP Demultiplexing