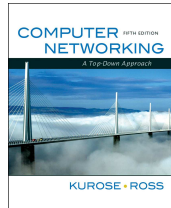


Chapter 2 Application Layer



A note on the use of these ppt slides:

We're making these slides freely available to all (faculty, students, readers). They're in PowerPoint form so you can add, modify, and delete slides (including this one) and slide content to suit your needs. They obviously represent a lot of work on our part. In return for use, we only ask the following:

- If you use these slides (e.g., in a class) in substantially unaltered form, that you mention their source (after all, we'd like people to use our book!)
- If you post any slides in substantially unaltered form on a www site, that you note that they are adapted from (or perhaps identical to) our slides, and note our copyright of this material.

Thanks and enjoy! JF/KWR

All material copyright 1996-2009
J.F. Kurose and K.W. Ross, All Rights Reserved

*Computer Networking:
A Top Down Approach,
5th edition.
Jim Kurose, Keith Ross
Addison-Wesley, April
2009.*

2: Application Layer 1

Chapter 2: Application layer

- 2.1 Principles of network applications
- 2.2 Web and HTTP
- 2.3 FTP
- 2.4 Electronic Mail
 - ❖ SMTP, POP3, IMAP
- 2.5 DNS
- 2.6 P2P applications
- 2.7 Socket programming with TCP
- 2.8 Socket programming with UDP

2: Application Layer 2

Chapter 2: Application Layer

Our goals:

- conceptual, implementation aspects of network application protocols
- ❖ transport-layer service models
- ❖ client-server paradigm
- ❖ peer-to-peer paradigm
- learn about protocols by examining popular application-level protocols
 - ❖ HTTP
 - ❖ FTP
 - ❖ SMTP / POP3 / IMAP
 - ❖ DNS
- programming network applications
 - ❖ socket API

2: Application Layer 3

Some network apps

- e-mail
- web
- instant messaging
- remote login
- P2P file sharing
- multi-user network games
- streaming stored video clips
- voice over IP
- real-time video conferencing
- grid computing
-
-

2: Application Layer 4

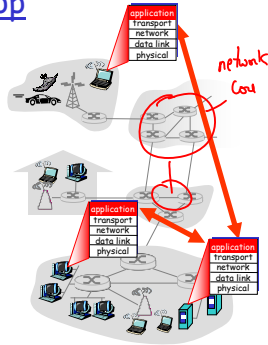
Creating a network app

write programs that

- ❖ run on (different) *end systems*
- ❖ communicate over network
- ❖ e.g., web server software communicates with browser software

No need to write software for network-core devices

- ❖ Network-core devices do not run user applications
- ❖ applications on end systems allows for rapid app development, propagation



Chapter 2: Application layer

2.1 Principles of network applications

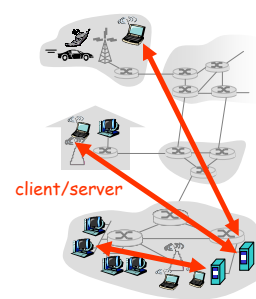
- ❑ 2.2 Web and HTTP
- ❑ 2.3 FTP
- ❑ 2.4 Electronic Mail
 - ❖ SMTP, POP3, IMAP
- ❑ 2.5 DNS

- ❑ 2.6 P2P applications
- ❑ 2.7 Socket programming with TCP
- ❑ 2.8 Socket programming with UDP
- ❑ 2.9 Building a Web server

Application architectures

- ❑ Client-server
- ❑ Peer-to-peer (P2P)
- ❑ Hybrid of client-server and P2P

Client-server architecture



passive server:

- ❖ always-on host
- ❖ permanent IP address
- ❖ server farms for scaling

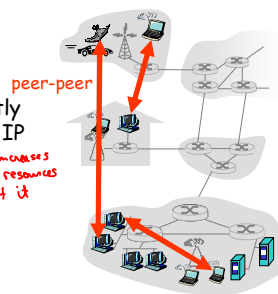
clients: initiate interaction

- ❖ communicate with server
- ❖ may be intermittently connected
- ❖ may have dynamic IP addresses
- ❖ do not communicate directly with each other

Pure P2P architecture

- no always-on server
- arbitrary end systems directly communicate
- peers are intermittently connected and change IP addresses

Highly scalable but difficult to manage



Hybrid of client-server and P2P

Skype

- voice-over-IP P2P application
- centralized server: finding address of remote party:
- client-client connection: direct (not through server)

Instant messaging

- chatting between two users is P2P
- centralized service: client presence detection/location
 - user registers its IP address with central server when it comes online
 - user contacts central server to find IP addresses of buddies

Processes communicating

- Process:** program running within a host.
- within same host, two processes communicate using **inter-process communication** (defined by OS). - message based shared memory
 - processes in different hosts communicate by exchanging **messages**

Client process: process that initiates communication

Server process: process that waits to be contacted

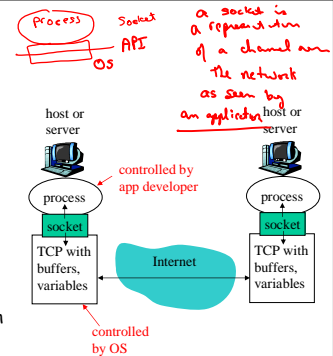
- Note: applications with P2P architectures have client processes & server processes

Sockets

Port ≠ socket

Sockets

- process sends/receives messages to/from its **socket**
- socket analogous to door
 - sending process shoves message out door
 - sending process relies on transport infrastructure on other side of door which brings message to socket at receiving process
- API: (1) choice of transport protocol; (2) ability to fix a few parameters (lots more on this later)



Addressing processes

- to receive messages, process must have *identifier*
- host device has unique 32-bit IP address
- Q:** does IP address of host suffice for identifying the process?

Addressing processes

- to receive messages, process must have *identifier*
- host device has unique 32-bit IP address
- Q:** does IP address of host on which process runs suffice for identifying the process?
 - A:** No, many processes can be running on same host
- identifier** includes both IP address and port numbers associated with process on host.
- Example port numbers:
 - HTTP server: 80 *well known ports*
 - Mail server: 25 *well known ports services*
- to send HTTP message to gaia.cs.umass.edu web server:
 - IP address: 128.119.245.12
 - Port number: 80
- more shortly...

App-layer protocol defines

- Types of messages exchanged,
 - e.g., request, response
 - Message syntax:
 - what fields in messages & how fields are delineated
 - Message semantics
 - meaning of information in fields
 - Rules for when and how processes send & respond to messages
- Public-domain protocols:**
- defined in RFCs
 - allows for interoperability
 - e.g., HTTP, SMTP
- Proprietary protocols:**
- e.g., Skype

What transport service does an app need?

- Data loss**
- some apps (e.g., audio) can tolerate some loss
 - other apps (e.g., file transfer, telnet) require 100% reliable data transfer
- Timing**
- some apps (e.g., Internet telephony, interactive games) require low delay to be "effective"
- Throughput**
- some apps (e.g., multimedia) require minimum amount of throughput to be "effective"
 - other apps ("elastic apps") make use of whatever throughput they get
- Security**
- Encryption, data integrity, ...

Transport service requirements of common apps

Application	Data loss	Throughput	Time Sensitive
file transfer	no loss	elastic	no
e-mail	no loss	elastic	no
Web documents	no loss	elastic	no
real-time audio/video	loss-tolerant	audio: 5kbps-1Mbps video: 10kbps-5Mbps	yes, 100's msec
stored audio/video	loss-tolerant	same as above	yes, few secs
interactive games	loss-tolerant	few kbps up	yes, 100's msec
instant messaging	no loss	elastic	yes and no

Internet transport protocols services

Transmission Control Protocol

TCP service:

- connection-oriented: setup required between client and server processes
- reliable transport between sending and receiving process
- flow control: sender won't overwhelm receiver
- congestion control: throttle sender when network overloaded
- does not provide: timing, minimum throughput guarantees, security

Unreliable Datagram Protocol

UDP service:

- unreliable data transfer between sending and receiving process
- does not provide: connection setup, reliability, flow control, congestion control, timing, throughput guarantee, or security

Little more than IP itself: adds port num and a little error checking

Q: why bother? Why is there a UDP?

Internet apps: application, transport protocols

Application	Application layer protocol	Underlying transport protocol
e-mail	SMTP [RFC 2821]	TCP
remote terminal access	Telnet [RFC 854]	TCP
Web	HTTP [RFC 2616]	TCP
file transfer	FTP [RFC 959]	TCP
streaming multimedia	HTTP (eg Youtube), RTP [RFC 1889]	TCP or UDP
Internet telephony	SIP, RTP, proprietary (e.g., Skype)	typically UDP

Chapter 2: Application layer

- 2.1 Principles of network applications
 - ✦ app architectures
 - ✦ app requirements
- 2.2 Web and HTTP
- 2.4 Electronic Mail
 - ✦ SMTP, POP3, IMAP
- 2.5 DNS
- 2.6 P2P applications
- 2.7 Socket programming with TCP
- 2.8 Socket programming with UDP

Web and HTTP

First some jargon

- Web page consists of objects
- Object can be HTML file, JPEG image, Java applet, audio file,...
- Web page consists of **base HTML-file** which includes several referenced objects
- Each object is addressable by a **URL**
- Example URL:

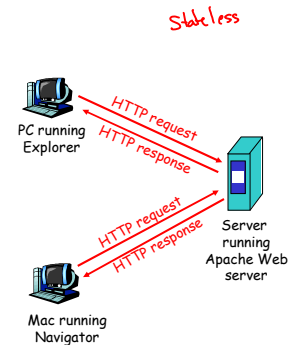
http : // www.someschool.edu / someDept/pic.gif

Handwritten notes:
 - http: protocol name
 - www.someschool.edu: host name (not enough)
 - someDept/pic.gif: path name
 - www: Uniform Resource Locators
 - someDept/pic.gif: location on host
 - http: identify a host protocol

HTTP overview

HTTP: hypertext transfer protocol

- Web's application layer protocol
- client/server model
 - client*: browser that requests, receives, "displays" Web objects
 - server*: Web server sends objects in response to requests



HTTP overview (continued)

Uses TCP:

- client initiates TCP connection (creates socket) to server, port 80
- server accepts TCP connection from client
- HTTP messages (application-layer protocol messages) exchanged between browser (HTTP client) and Web server (HTTP server)
- TCP connection closed

HTTP is "stateless"

Handwritten note: - simplifies

- server maintains no information about past client requests

aside

Protocols that maintain "state" are complex!

- past history (state) must be maintained
- if server/client crashes, their views of "state" may be inconsistent, must be reconciled

HTTP connections

Nonpersistent HTTP

- At most one object is sent over a TCP connection.

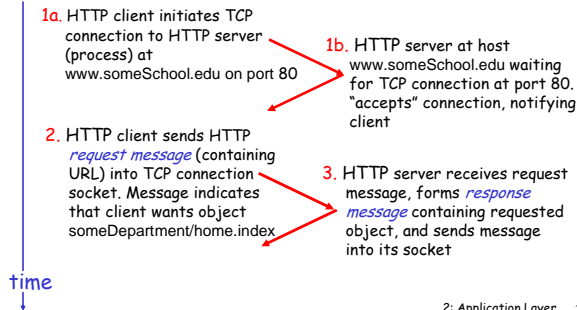
Persistent HTTP

- Multiple objects can be sent over single TCP connection between client and server.

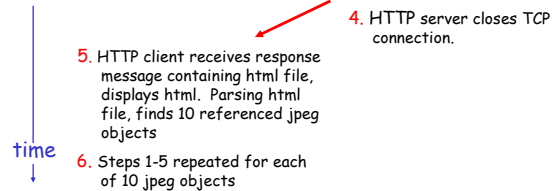
Nonpersistent HTTP

Suppose user enters URL

www.someSchool.edu/someDepartment/home.index (contains text, references to 10 jpeg images)



Nonpersistent HTTP (cont.)

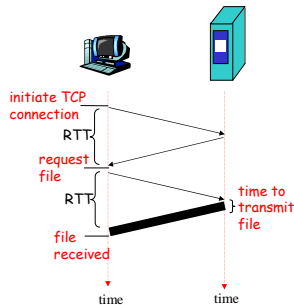


Non-Persistent HTTP: Response time

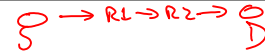
Definition of RTT: time for a small packet to travel from client to server and back.

Response time:

- one RTT to initiate TCP connection
 - one RTT for HTTP request and first few bytes of HTTP response to return
 - file transmission time
- total = 2RTT + transmit time**



Persistent HTTP

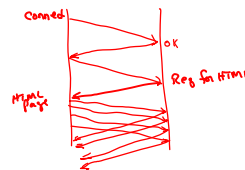


Nonpersistent HTTP issues:

- requires 2 RTTs per object
- OS overhead for each TCP connection
- browsers often open parallel TCP connections to fetch referenced objects

Persistent HTTP

- server leaves connection open after sending response
- subsequent HTTP messages between same client/server sent over open connection
- client sends requests as soon as it encounters a referenced object
- as little as one RTT for all the referenced objects



HTTP request message

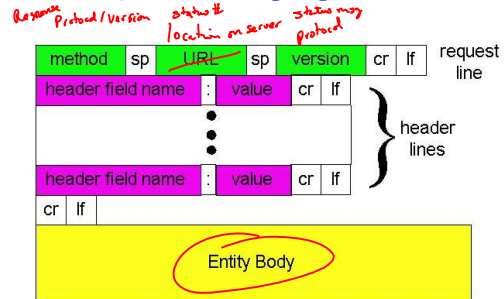
- two types of HTTP messages: *request, response*
- HTTP request message:
 - ASCII (human-readable format)

request line (GET, POST, HEAD commands) → GET /somedir/page.html HTTP/1.1

header lines → Host: www.someschool.edu
User-agent: Mozilla/4.0
Connection: close
Accept-language: fr

Carriage return line feed (extra carriage return, line feed) indicates end of message →

HTTP request message: general format



Requests for Uploading form input

POST Post method:

- Web page often includes form input
- Input is uploaded to server in entity body

GET URL method:

- Uses GET method
- Input is uploaded in URL field of request line:

http://www.somesite.com/animalsearch?monkeys&banana

Method types

HTTP/1.0

- GET
- POST
- HEAD
 - asks server to leave requested object out of response

HTTP/1.1

- GET, POST, HEAD
- PUT
 - uploads file in entity body to path specified in URL field
- DELETE
 - deletes file specified in the URL field

HTTP response message

status line
(protocol
status code
status phrase) → HTTP/1.1 200 OK
Connection: close
Date: Thu, 06 Aug 1998 12:00:15 GMT
Server: Apache/1.3.0 (Unix)
Last-Modified: Mon, 22 Jun 1998
Content-Length: 6821
Content-Type: text/html

header lines

data, e.g.,
requested
HTML file → data data data data data ...

HTTP response status codes

In first line in server→client response message.
A few sample codes:

- 200 OK**
 - ✦ request succeeded, requested object later in this message
- 301 Moved Permanently**
 - ✦ requested object moved, new location specified later in this message (Location:)
- 400 Bad Request**
 - ✦ request message not understood by server
- 404 Not Found**
 - ✦ requested document not found on this server
- 505 HTTP Version Not Supported**

Trying out HTTP (client side) for yourself

1. Telnet to your favorite Web server:

```
telnet cis.poly.edu 80
```

Opens TCP connection to port 80 (default HTTP server port) at cis.poly.edu. Anything typed in sent to port 80 at cis.poly.edu

2. Type in a GET HTTP request:

```
GET /~ross/ HTTP/1.1  
Host: cis.poly.edu
```

By typing this in (hit carriage return twice), you send this minimal (but complete) GET request to HTTP server

3. Look at response message sent by HTTP server!