Application Layer Protocols

Moustafa Youssef

Layered Architecture

- Why layering?
- Independent design of each layer
- Need to agree on the interface.

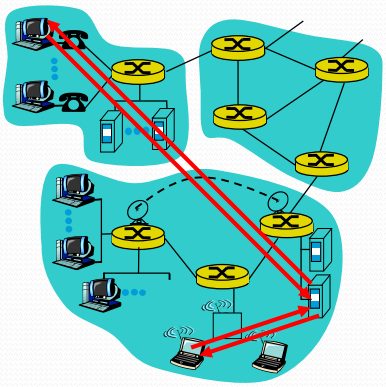
Some network apps

- E-mail
- Web
- Instant messaging
- Remote login
- P2P file sharing
- Multi-user network games
- Streaming stored video clips
- Internet telephone
- Real-time video conference
- Massive parallel computing
- Cloud computing
-
-

Application architectures

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

Client-server architecture



server:

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

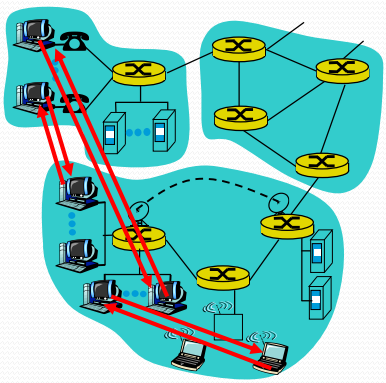
clients:

- 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
- example: Gnutella

Highly scalable but difficult to manage



Hybrid of client-server and P2P

Skype

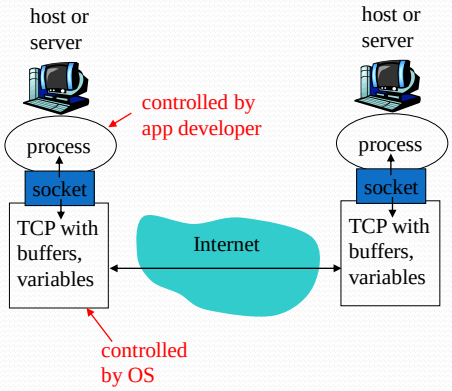
- Internet telephony app
- Finding address of remote party: centralized server(s)
- Client-client connection is direct (not through server)

Instant messaging

- Chatting between two users is P2P
- Presence detection/location centralized:
  - User registers its IP address with central server when it comes online
  - User contacts central server to find IP addresses of buddies

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 unique32-bit IP address
- Q: does IP address of host on which process runs suffice for identifying the process?

Addressing processes

- to receive messages, process must have *identifier*
- host device has unique32-bit IP address
- Q: does IP address of host on which process runs suffice for identifying the process?
  - Answer: 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
  - Mail server: 25
- 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, KaZaA

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"

Bandwidth

- some apps (e.g., multimedia) require minimum amount of bandwidth to be "effective"
- other apps ("elastic apps") make use of whatever bandwidth they get

Transport service requirements of common apps

Application	Data loss	Bandwidth	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

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 bandwidth guarantees

UDP service:

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

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	proprietary (e.g. RealNetworks)	TCP or UDP
Internet telephony	proprietary (e.g., Vonage,Dialpad)	typically UDP

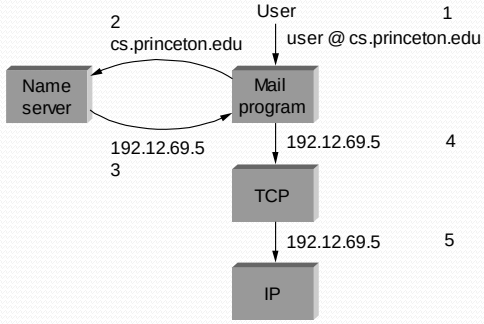
Application Layer Protocols

- DNS
- HTTP

What is a DNS Service

- Maps from host names to IP addresses
- Advantages of host names
  - Easily recognizable by humans
- Disadvantages
  - Hard for computers, why?
  - No information to locate host
- Distributed DB

Example: DNS Resolution



Terminology

- Name space: set of possible names
  - Can be flat or hierarchical
  - File system hierarchy
- Binding; mapping from names to values
- Resolution mechanism: when invoked with a name, returns a value
  - Name server

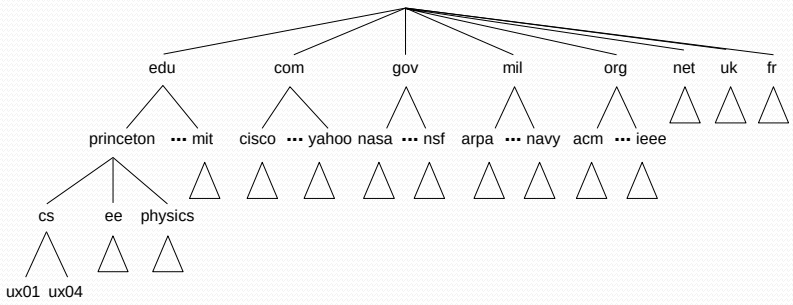
Early Implementation

- Hosts.txt file
  - Kept at the Network Information Center (NIC)
  - Sent by email every few days
  - Manually updated by administrators
- Why not centralized?

Current Implementation

- Distributed hierarchical name space
- Example
  - www.alex.edu.eg
- Read from left
- Processed from right

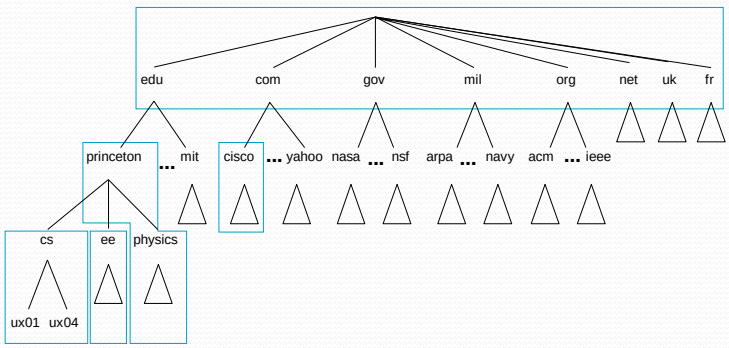
Example: DNS Hierarchy



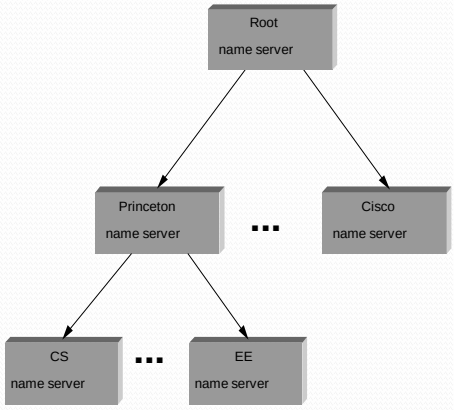
Name Servers

- Zone: a subtree that belongs to one administrative domain
- At least two servers per zone
  - Why?
- One server can handle more than one zone

Example: Zones



Example: Hierarchy of Mail Servers



Name Servers' Records

- DNS: distributed db storing resource records (RR)

RR format: **`(name, value, type, Class,ttl)`**

❒ Type=A
- ❖ **`name`** is hostname
- ❖ **`value`** is IP address

- Type=NS
  - **`name`** is domain (e.g. foo.com)
  - **`value`** is hostname of authoritative name server for this domain

❒ Type=CNAME
- ❖ **`name`** is alias name for some "canonical" (the real) name `www.ibm.com` is really `servereast.backup2.ibm.com`
- ❖ **`value`** is canonical name

❒ Type=MX
- ❖ **`value`** is name of mailserver associated with **`name`**

Examples

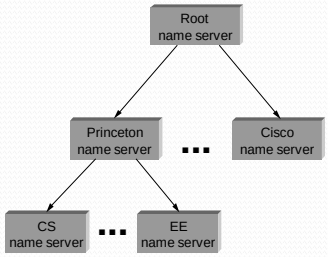
<princeton.edu, cit.princeton.edu, NS, IN>
<cit.princeton.edu, 128.196.128.233, A, IN>

<cisco.com, ns.cisco.com, NS, IN>
<ns.cisco.com, 128.96.32.20, A, IN>

Examples

<cs.princeton.edu,gnat.cs.princeton.edu,MX,IN>
<cicada.cs.princeton.edu,192.12.69.60,A,IN>
<cic.cs.princeton.edu,cicada.cs.princeton.edu,CNAME,IN>
<www.cs.princeton.edu,192.12.69.35,A,IN>
<cicada.cs.princeton.edu,roach.cs.princeton.edu,CNAME,IN>

Name Resolution



- What's a DNS client, Where it's?
- Resolve
  - cicada.cs.princeton.edu
- Root server

<princeton.edu, cit.princeton.edu, NS, IN>

<cit.princeton.edu, 128.196.128.233, A, IN>

- cit.princeton.edu

<cs.princeton.edu, gnat.cs.princeton.edu, NS, IN>

<gnat.cs.princeton.edu, 192.12.69.5, A, IN>

- gnat.cs.princeton.edu returns required IP

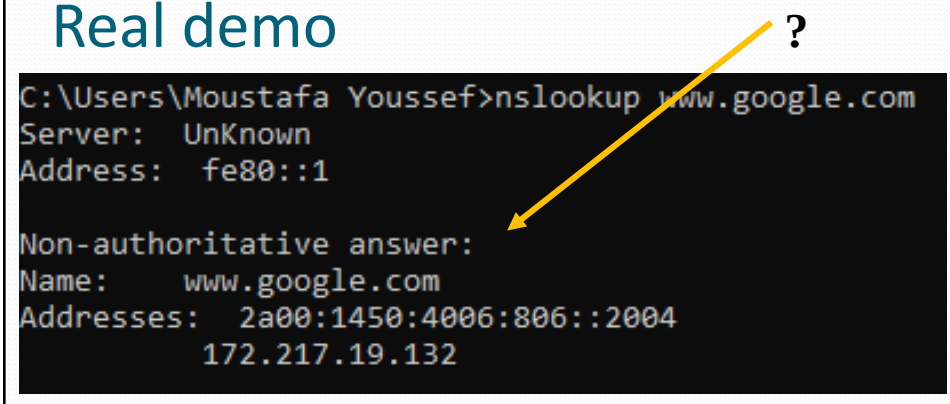
Required to be resolved: www.cs.princeton.edu

Real demo

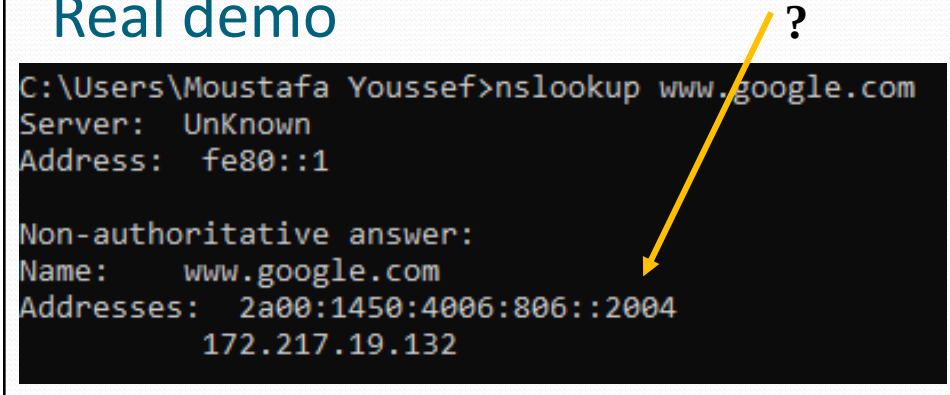
```
C:\Users\Moustafa Youssef>nslookup www.google.com
Server:  UnKnown
Address:  fe80::1

Non-authoritative answer:
Name:    www.google.com
Addresses:  2a00:1450:4006:806::2004
          172.217.19.132
```

Real demo



Real demo



```
C:\Users\Moustafa Youssef>nslookup
Default Server:  UnKnown
Address:  fe80::1

> set debug
> www.google.com
Server:  UnKnown
Address:  fe80::1

------------
Got answer:
    HEADER:
        opcode = QUERY, id = 2, rcode = NXDOMAIN
        header flags:  response, want recursion, recursion avail.
        questions = 1,  answers = 0,  authority records = 0,  additional = 1

    QUESTIONS:
        www.google.com.home, type = A, class = IN
    ADDITIONAL RECORDS:
    ->  tunneling
        ttl = 900 (15 mins)
        primary name server = auto-dns01-c-eg.te
        responsible mail addr = please_set_email.absolutely.nowhere
        serial  = 4924
        refresh = 10800 (3 hours)
        retry   = 3600 (1 hour)
        expire  = 2419200 (28 days)
        default TTL = 900 (15 mins)

------------
------------
Got answer:
    HEADER:
        opcode = QUERY, id = 3, rcode = NXDOMAIN
        header flags:  response, want recursion, recursion avail.
        questions = 1,  answers = 0,  authority records = 0,  additional = 1

    QUESTIONS:
        www.google.com.home, type = AAAA, class = IN
    ADDITIONAL RECORDS:
    ->  tunneling
        ttl = 900 (15 mins)
        primary name server = auto-dns01-c-eg.te
        responsible mail addr = please_set_email.absolutely.nowhere
```

Real demo

```
C:\Users\Moustafa Youssef>ipconfig /all

Windows IP Configuration

Wireless LAN adapter Wi-Fi:

   Connection-specific DNS Suffix  . : home
   Description . . . . . . . . . . . : Intel(R) Dual Band Wireless-AC 8265
   Physical Address. . . . . . . . . : 94-B8-6D-5B-35-7E
   DHCP Enabled. . . . . . . . . . . : Yes
   Autoconfiguration Enabled . . . . : Yes
   IPv6 Address. . . . . . . . . . . : 2600:1700:1150:dd30::48(Preferred)
   Lease Obtained. . . . . . . . . . : Monday, September 16, 2019 6:14:14 AM
   Lease Expires . . . . . . . . . . : Saturday, November 8, 2155 3:20:15 AM
   IPv6 Address. . . . . . . . . . . : fd1c:599b:90b6:7100:9481:42ed:e47c:8dbd(Preferred)
   Temporary IPv6 Address. . . . . . : fd1c:599b:90b6:7100:cce5:540a:5785:b414(Preferred)
   Link-local IPv6 Address . . . . . : fe80::9481:42ed:e47c:8dbd%3(Preferred)
   IPv4 Address. . . . . . . . . . . : 192.168.1.4(Preferred)
   Subnet Mask . . . . . . . . . . . : 255.255.255.0
   Lease Obtained. . . . . . . . . . : Tuesday, October 1, 2019 5:26:36 PM
   Lease Expires . . . . . . . . . . : Wednesday, October 2, 2019 7:54:07 PM
   Default Gateway . . . . . . . . . : 192.168.1.1
   DHCP Server . . . . . . . . . . . : 192.168.1.1
   DHCPv6 IAID . . . . . . . . . . . : 43300973
   DHCPv6 Client DUID. . . . . . . . : 00-01-00-01-23-2A-8F-7D-8C-16-45-9F-FA-A2
   DNS Servers . . . . . . . . . . . : fe80::1%3
                                       208.67.222.222
                                       208.67.220.220
   NetBIOS over Tcpip. . . . . . . . : Enabled
```

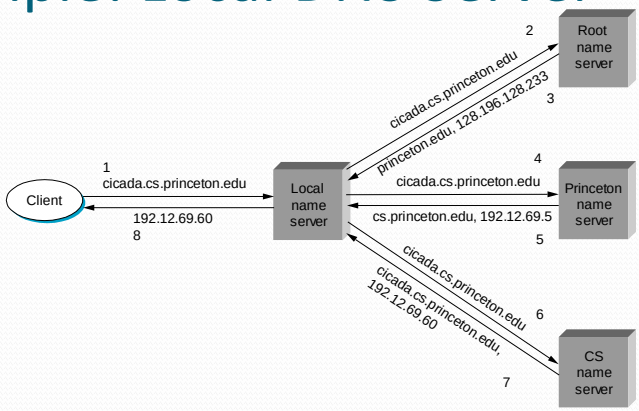
What's Missing?

- Bootstrapping
- Submitting partial names

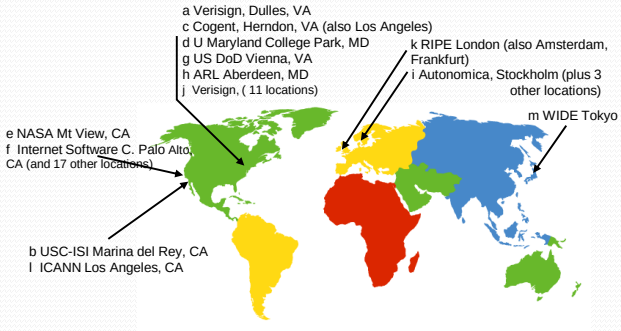
Bootstrapping

- Store root server IP
- Use local name server
- Not necessarily part of the DNS hierarchy
- Can use recursive resolving
- How to setup your DNS name server
  - Show properties of TCP tab

Example: Local DNS Server



Root name servers



13 root name servers worldwide

To Do

- Read DNS RFC
- Mapping from IP to physical address (e.g., MAC)?
- Get your local DNS IP address
- Get the IP address of google.com
  - Use nslookup, web versions available
- Read: Mockapetris, Development of the Domain Name System, SIGCOMM 88.
- Read: Ramasubramanian and Sirer, The Design and Implementation of a Next Generation Name Service for the Internet. SIGCOMM 04.

HTTP

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:

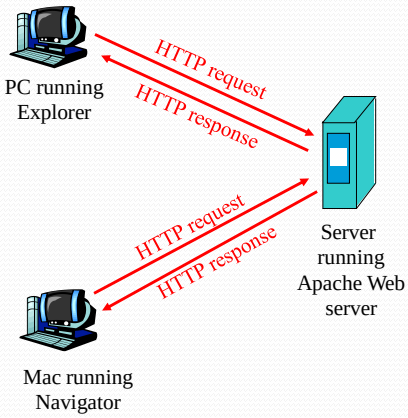
`www.someschool.edu/someDept/pic.gif`

host name | path name

HTTP overview

HTTP: hypertext transfer protocol

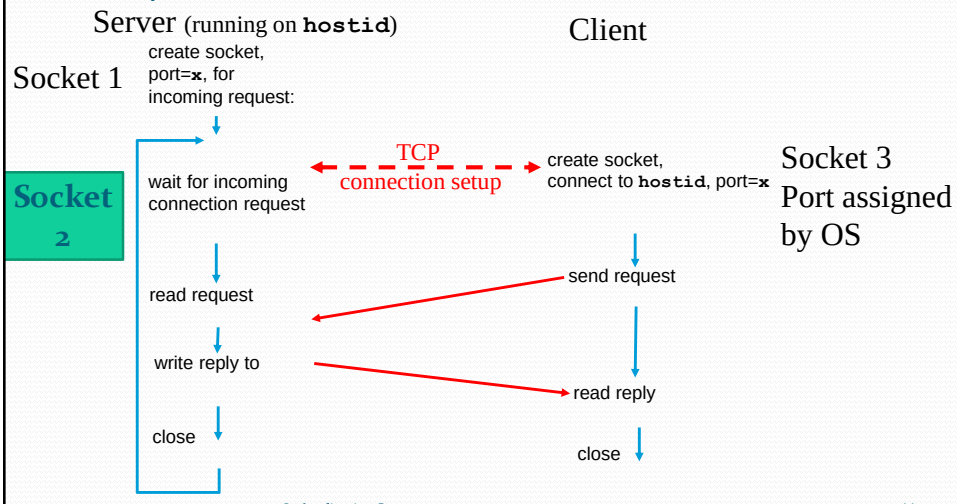
- Web's application layer protocol
- client/server model
  - *client:* browser that requests, receives, "displays" Web objects
  - *server:* Web server sends objects in response to requests
- HTTP 1.0: RFC 1945
- HTTP 1.1: RFC 2068



Question

- How to have one single port
  - When you have multiple different clients connecting to the same server
  - May be from the same machine

Client/server socket interaction: TCP



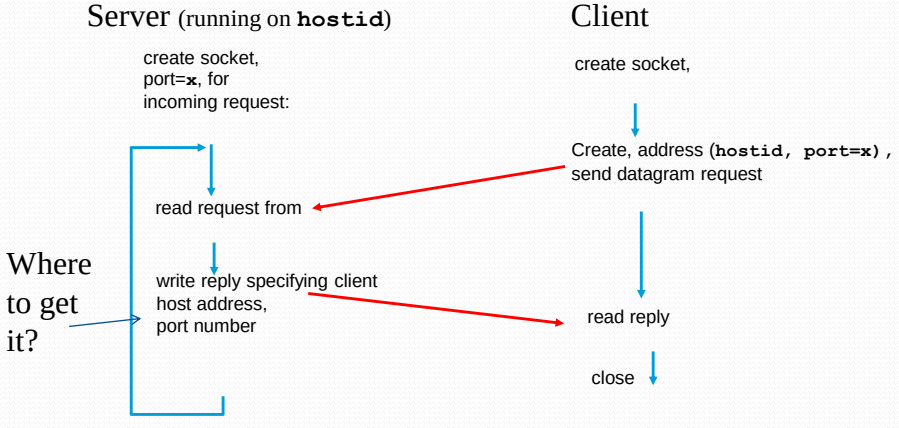
Socket programming *with UDP*

UDP: no "connection" between client and server

- no handshaking
- sender explicitly attaches IP address and port of destination to each packet
- server must extract IP address, port of sender from received packet

UDP: transmitted data may be received out of order, or lost

Client/server socket interaction: UDP



HTTP Summary

- Uses TCP
- Stateless
  - Advantages
- Persistent vs. non-persistent connections
  - Pipeline vs. no pipeline
- Caching

Example: An HTTP Request

- **GET / HTTP/1.0**
- **Empty line ☹**

- **Where's www.google.com?**
- **Where is the URL?**
- **www.cs.umd.edu:80/~moustafa**
- **http://www.cs.umd.edu:80/~moustafa**

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"

- 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.
- HTTP/1.0 uses nonpersistent HTTP

Persistent HTTP

- Multiple objects can be sent over single TCP connection between client and server.
- HTTP/1.1 uses persistent connections in default mode

Nonpersistent HTTP

Suppose user enters URL `www.someSchool.edu/someDepartment/home.index` (contains text, references to 10 jpeg images)

1a. HTTP client initiates TCP connection to HTTP server (process) at www.someSchool.edu on port 80

1b. HTTP server at host www.someSchool.edu waiting for TCP connection at port 80. "accepts" connection, notifying client

2. HTTP client sends HTTP *request message* (containing URL) into TCP connection socket. Message indicates that client wants object someDepartment/home.index

3. HTTP server receives request message, forms *response message* containing requested object, and sends message into its socket

time

Nonpersistent HTTP (cont.)

4. HTTP server closes TCP connection.

5. HTTP client receives response message containing html file, displays html. Parsing html file, finds 10 referenced jpeg objects

time

6. Steps 1-5 repeated for each of 10 jpeg objects

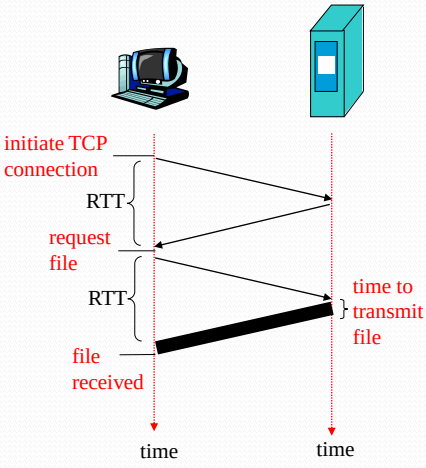
Non-Persistent HTTP: Response time

Definition of RTT: time to send 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

Persistent ***without*** pipelining:

- client issues new request only when previous response has been received
- one RTT for each referenced object

Persistent ***with*** pipelining:

- default in HTTP/1.1
- 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)

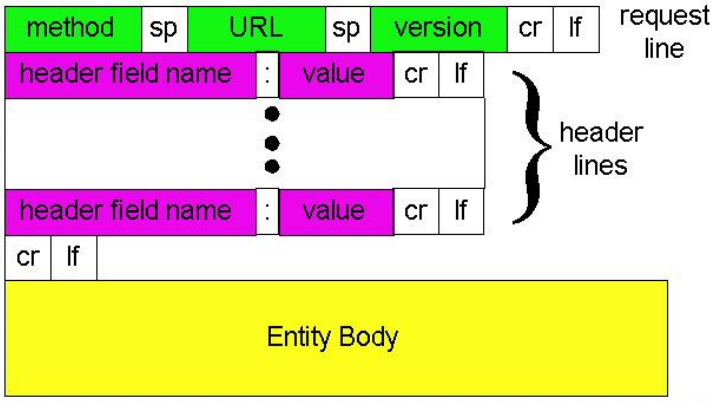
header lines

```
GET /somedir/page.html HTTP/1.1
Host: www.someschool.edu
User-agent: Mozilla/4.0
Connection: close
Accept-language:fr
```

Carriage return, line feed indicates end of message

(extra carriage return, line feed)

HTTP request message: general format



Uploading form input

Post method:

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

URL method:

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

```
www.somesite.com/animalsearch?monkeys&banana
```


Method types

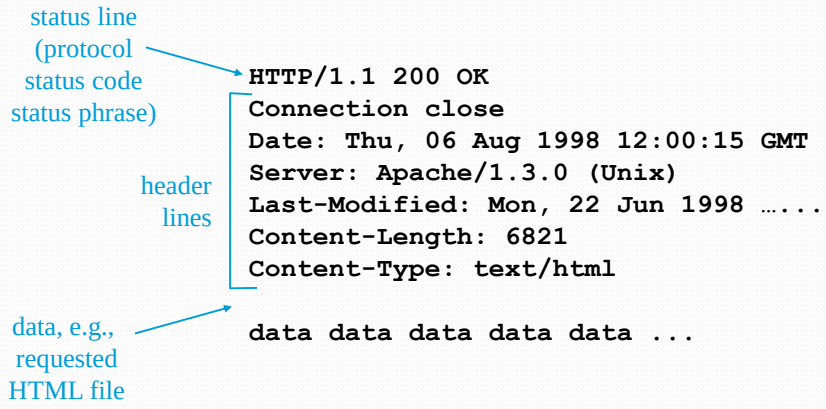
HTTP/1.0

- GET
- POST
- HEAD
  - Just returns the headers
  - Asks server to leave requested object out of response
  - Useful e.g. to decide whether to download a file or not based on size

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



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!

Let's look at HTTP in action

- telnet example
- Wireshark example

User-server state: cookies

Many major Web sites use cookies

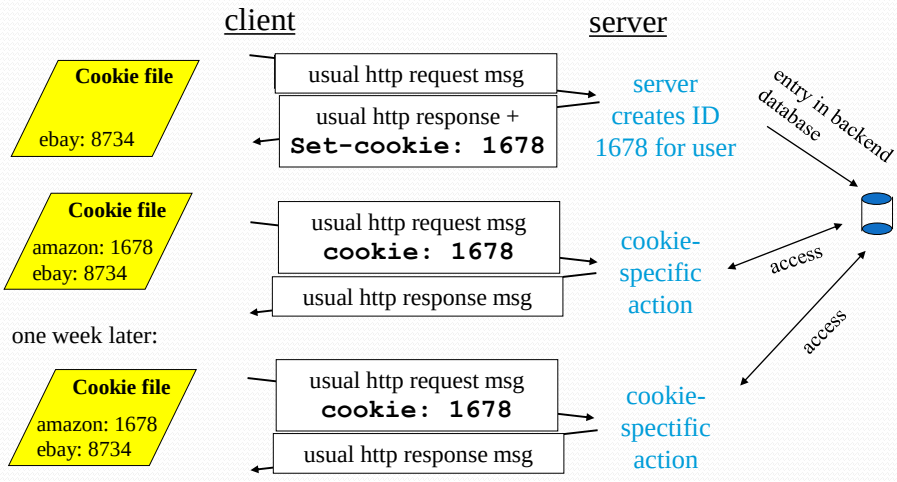
Four components:

1) cookie header line of HTTP *response* message
2) cookie header line in HTTP *request* message
3) cookie file kept on user's host, managed by user's browser
4) back-end database at Web site

Example:

- Susan access Internet always from same PC
- She visits a specific e-commerce site for first time
- When initial HTTP requests arrives at site, site creates a unique ID and creates an entry in backend database for ID

Cookies: keeping "state" (cont.)



Cookies (continued)

What cookies can bring:

- authorization
- shopping carts
- recommendations
- user session state (Web e-mail)

aside

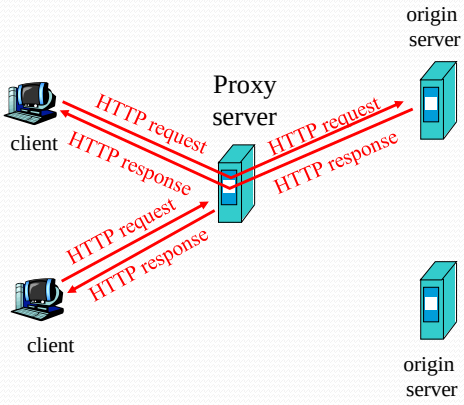
Cookies and privacy:

- ❒ cookies permit sites to learn a lot about you
- ❒ you may supply name and e-mail to sites
- ❒ search engines use redirection & cookies to learn yet more
- ❒ advertising companies obtain info across sites

Web caches (proxy server)

Goal: satisfy client request without involving origin server

- user sets browser: Web accesses via cache
- browser sends all HTTP requests to cache
  - object in cache: cache returns object
  - else cache requests object from origin server, then returns object to client



More about Web caching

- Cache acts as both client and server
- Typically cache is installed by ISP (university, company, residential ISP)

Why Web caching?

- Reduce response time for client request.
- Reduce traffic on an institution's access link.
- Internet dense with caches enables "poor" content providers to effectively deliver content (but so does P2P file sharing)

Caching example

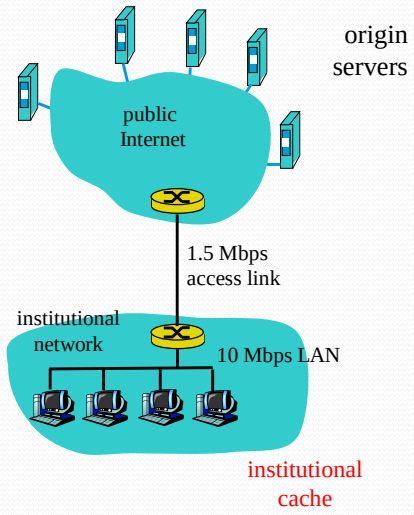
Assumptions

- average object size = 100,000 bits
- avg. request rate from institution's browsers to origin servers = 15/sec
- delay from institutional router to any origin server and back to router = 2 sec

Consequences

- utilization on LAN = 15%
- utilization on access link = 100%
- total delay = Internet delay + access delay + LAN delay

= 2 sec + minutes + milliseconds



Caching example (cont)

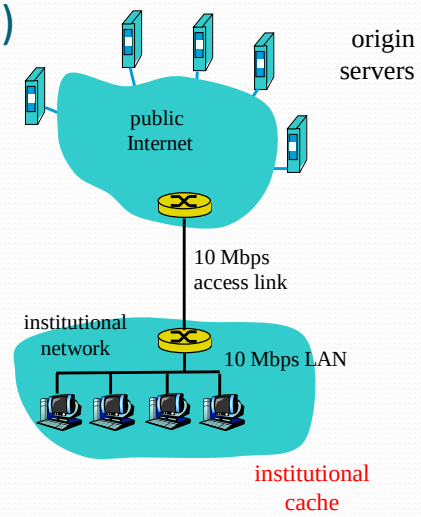
Possible solution

- increase bandwidth of access link to, say, 10 Mbps

Consequences

- utilization on LAN = 15%
- utilization on access link = 15%
- Total delay = Internet delay + access delay + LAN delay

  = 2 sec + msecs + msecs
- often a costly upgrade



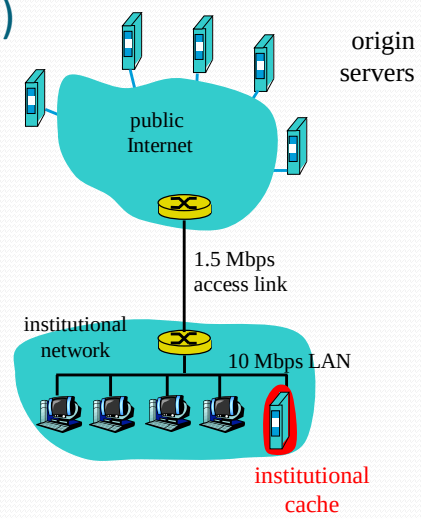
Caching example (cont)

Install cache

- suppose hit rate is .4

Consequence

- 40% requests will be satisfied almost immediately
- 60% requests satisfied by origin server
- utilization of access link reduced to 60%, resulting in negligible delays (say 10 msec)
- total avg delay = Internet delay + access delay + LAN delay = .6*(2.01) secs + .4*milliseconds < 1.4 secs

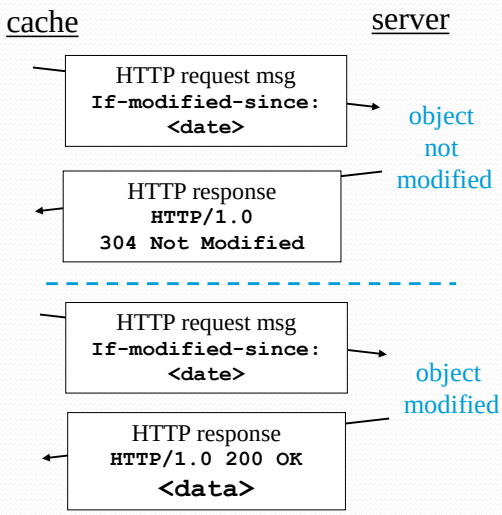


Conditional GET

- Goal: don't send object if cache has up-to-date cached version
- cache: specify date of cached copy in HTTP request

  `If-modified-since: <date>`
- server: response contains no object if cached copy is up-to-date:

  `HTTP/1.0 304 Not Modified`



To Do

- Read other applications layer protocols from the book
- Use telnet and Wireshark to test HTTP. See textbook
- Look into HTML page source
  - Is it related to HTTP
- Read HTTP RFCs