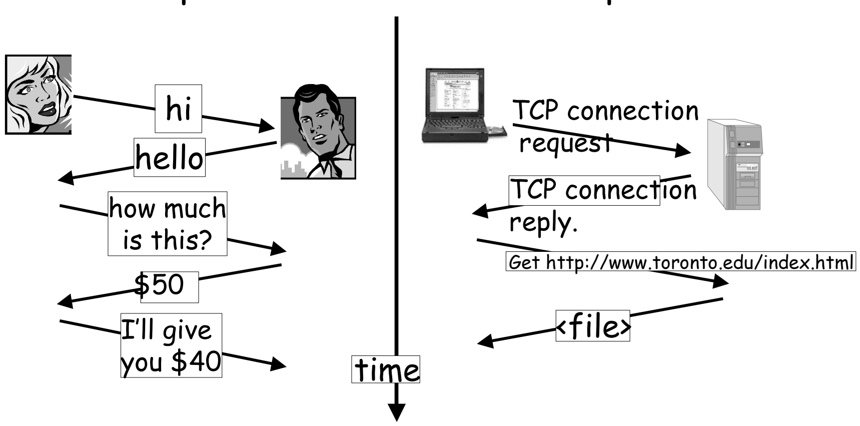
Protocols and HTTP, Web Browsers and Servers, Caching, DNS

first some background on
network-based applications …

What's a protocol?

human protocol network protocol:



What's a protocol?

Human Protocols:

- "thank you … you're welcome"
- "hello … hi … my name is … pleased to meet you"
- Price haggling

… specific msgs sent

… specific actions taken when msgs received

… may be context or culture sensitive

Network protocols:

- drive device, rather than human, interaction
- all communication activity in Internet is governed by protocols

protocols define format & order of messages sent and received among network entities, and actions taken on message transmission, receipt

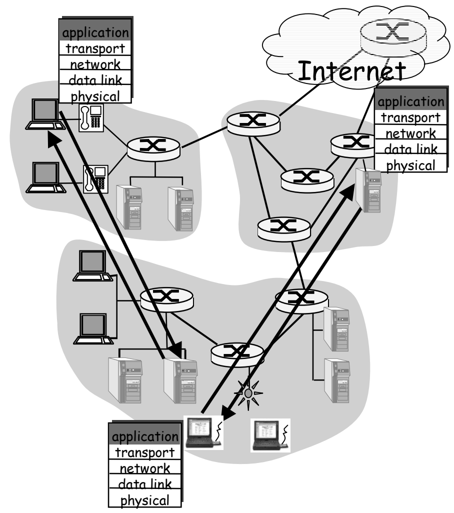
Applications and application-layer protocols

Application: communicating, distributed processes

- running in network hosts in "user space"
- exchange messages to implement app
- e.g., email, file transfer, the Web

Application-layer protocols

- one "piece" of an app
- define messages exchanged by apps and actions taken
- user services provided by lower layer protocols



Network applications: some jargon

- A process is a program that is running within a host.
- Within the same host, two processes communicate with inter-process communication defined by the OS.
- Processes running in different hosts communicate with an application-layer protocol
- A user agent is an interface between the user and the network application.
  - Web:browser
  - E-mail: mail reader
  - streaming audio/video: media player

Client-Server Paradigm

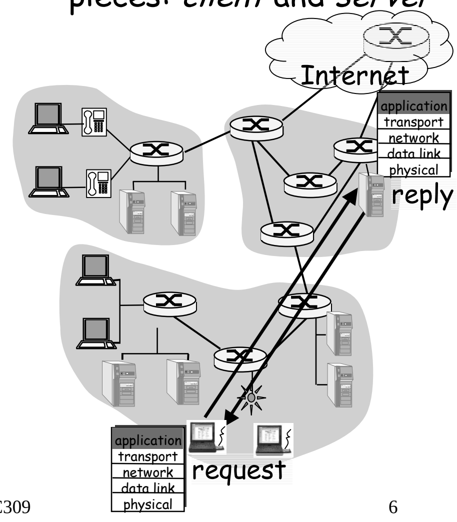
Client:

- initiates contact with server ("speaks first")
- typically requests service from server,
- for Web, client is implemented in browser; for e-mail, in mail reader

Server:

- provides requested service to client
- e.g., Web server sends requested Web page, mail server delivers e-mail

Typical network app has two pieces: *client* and *server*

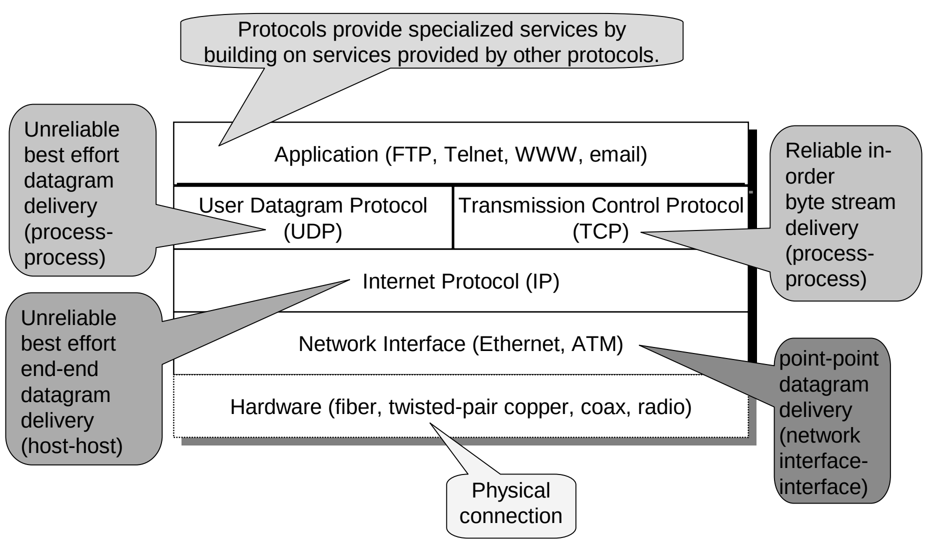


Application-layer protocols (cont).

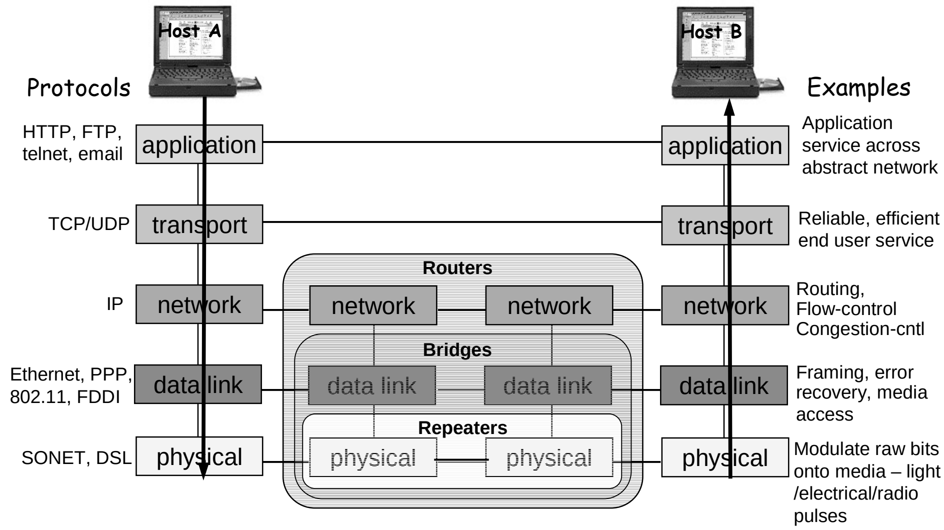
API: application programming interface
- defines interface between application and transport layer
- socket: Internet API
  - two processes communicate by sending data into socket, reading data out of socket

Q: how does a process "identify" the other process with which it wants to communicate?
- IP address of host running other process
- "port number" - allows receiving host to determine to which local process the message should be delivered

Protocol layering

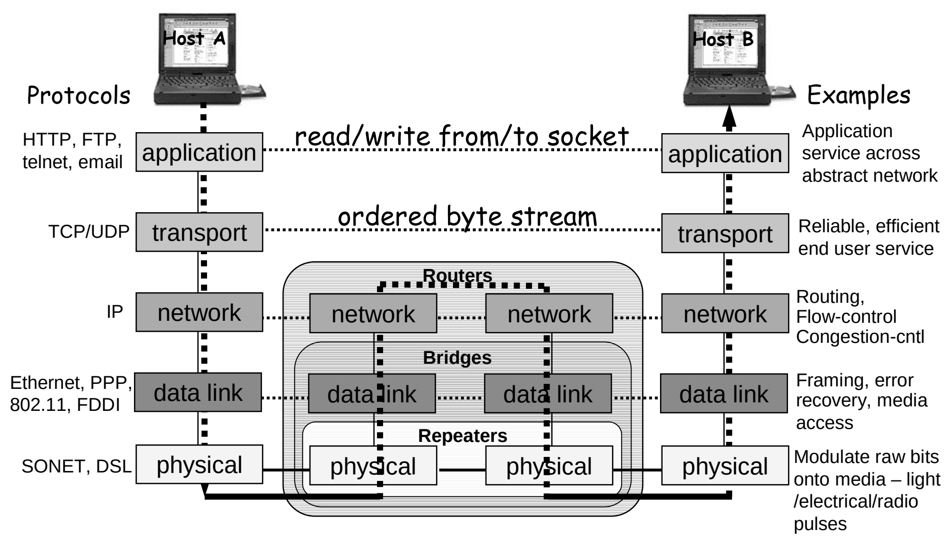


Protocol stacks



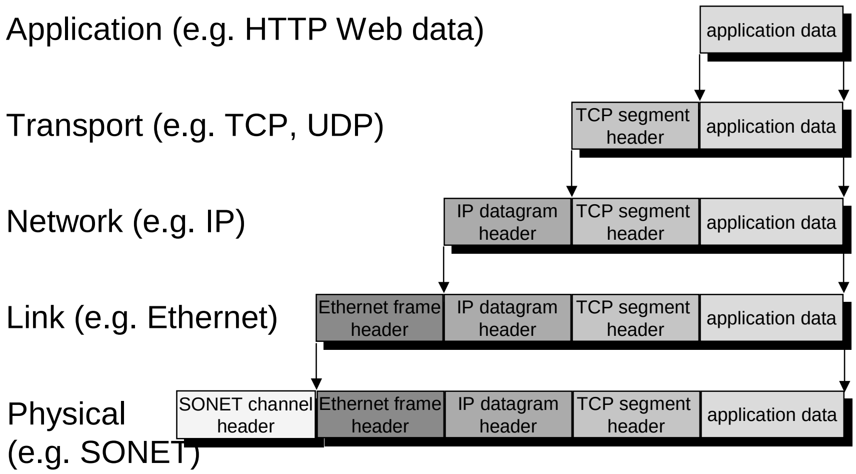
30 - http CSC309 9

Protocol Stacks



30 - http CSC309 10

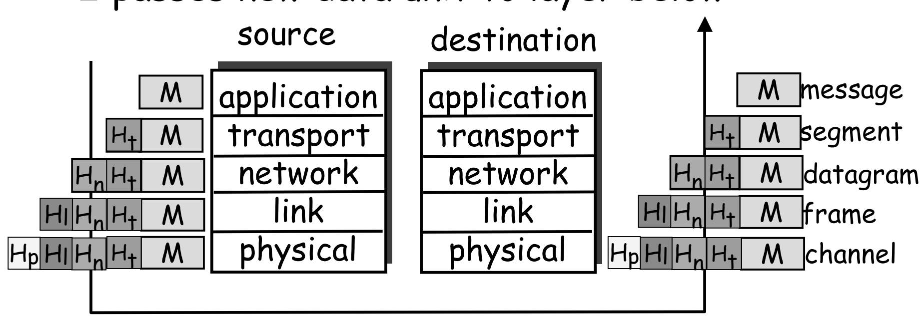
Encapsulation



Protocol layering and data

Each layer takes data from above and:

- adds header information to create new data unit
- passes new data unit to layer below



HTTP

- HyperText Transfer Protocol
- Created by Tim Berners-Lee at CERN
  - Physicists, not Computer Scientists
  - Share results from physics experiments
  - Defined 1989-1991
- Standardized and much expanded by the IETF
- Rides on top of TCP protocol
  - TCP provides: reliable, bi-directional, in-order byte stream
- Goal: transfer objects between systems
  - Do not confuse with other WWW concepts:
    - HTTP is not page layout language (that is HTML)
    - HTTP is not object naming scheme (that is URLs)

The Web: some jargon

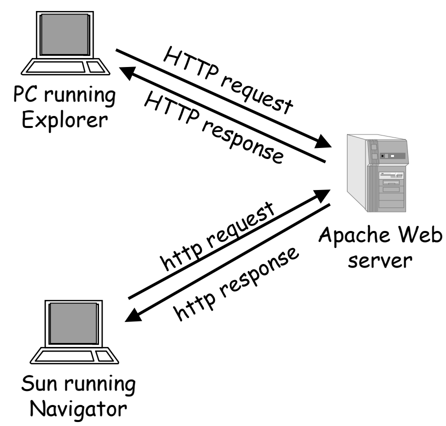
- Web page:
  - consists of "objects"
  - addressed by a URL
- Most Web pages consist of:
  - base HTML page
  - one or more referenced objects such as images
- URL has two components: host name and path name: e.g.
- User agent for Web is called a browser:
  - MS Internet Explorer
  - Netscape Communicator
- Server for Web is called Web server:
  - Apache (public domain)
  - MS Internet Information Server

`www.toronto.edu/depts/cs/pratt.jpg`

The Web: HTTP protocol

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
- http1.0: RFC 1945
- http1.1: RFC 2068



HTTP protocol (cont)

HTTP rides on top of TCP transport service:

- 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 protocol (cont)

http is "stateless"

- server maintains no information about past client requests

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 in operation

Suppose user enters URL
www.toronto.edu/cs/index.html (containing text and references to 10 jpeg images)

1a. http client initiates TCP connection to http server (process) at www.toronto.edu. Port 80 is default for http server.

1b. http server at host www.toronto.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

3. http server receives request message, forms *response message* containing requested object (/cs/index.html), sends message into socket

time

http in operation (cont.)

time

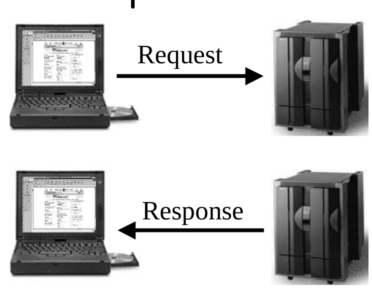
5. http client receives response message containing html file, displays html. Parsing html file, finds 10 referenced jpeg objects

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

4. http server closes TCP connection.

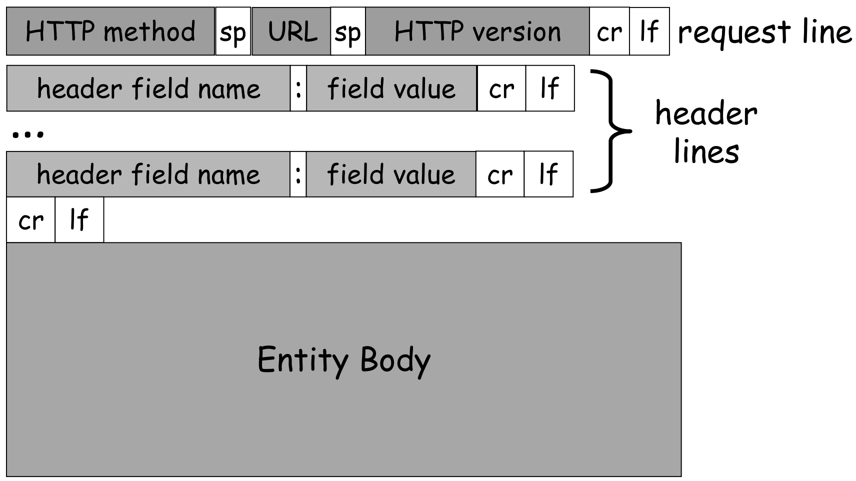
HTTP 1.0

- Interaction between Web client (browser) and Web server occurs in two phases:
  - Request phase: Browser requests page from Web server
  - Response phase: Server sends back requested page or code



- Each phase consists of two parts:
  - Header (request line, response status, header fields)
  - Body

HTTP request message: general format



http message format: request

- http request message:
  - Note all HTTP messages are in ASCII (text format)

```
GET /somedir/page.html HTTP/1.0
User-agent: Mozilla/4.0
Accept: text/html, image/gif,image/jpeg
Accept-language:fr
```

(extra carriage return, line feed)

request line (GET, POST, HEAD commands)

header lines

Carriage return, line feed indicates end of message

HTTP 1.0 Request Phase

- Request line

HTTP method	sp	URL	sp	HTTP version	cr	lf

  - HTTP method
    - GET – return content of specified document
    - HEAD – return headers only of GET response
    - POST – execute specified doc with enclosed data
  - URL (only domain portion)
    - /host-identifier/path
    - e.g. /www.toronto.edu/headlines/
  - HTTP version
    - e.g. HTTP/1.0

HTTP 1.0 Request Phase (cont)

- Header fields

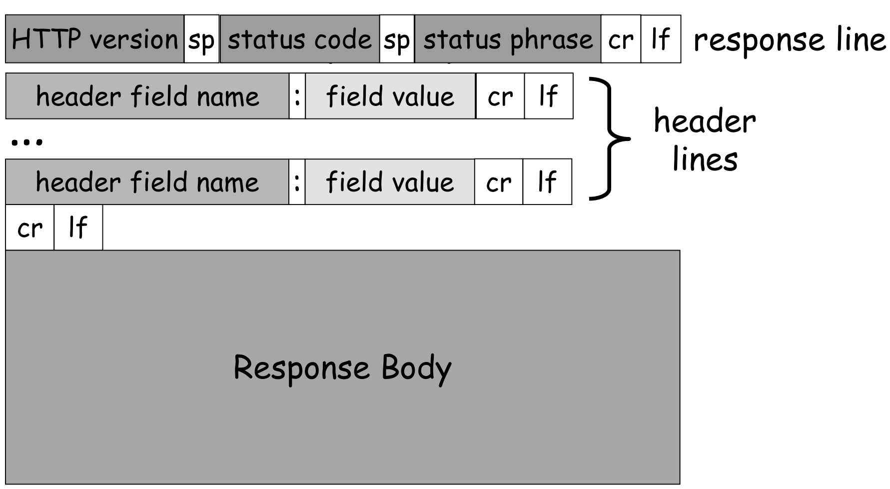
header field name	:	field value

- Examples:
  - Accept: text/html
  - Accept: image/jpg
  - Accept-language: en; en-gr; fr
  - If-modified-since: 17 May 2001
  - Content-Length: 2540

HTTP 1.0 Response Phase

- Response consists of
  - Status line:
    - HTTP version
    - 3-digit status code (success, error, redirection, etc)
    - Brief text explanation of status code (e.g. OK)
  - Response Header fields:
    - Other page attributes (content type, content length, expiration, last modified, server type, etc)
    - Additional information (if redirection, other location)
  - blank line (delimiter between header and body)
  - Response body

HTTP response message: general format



Response status codes

- 3 digit response code
  - 1XX – informational
  - 2XX – success
  - 3XX – redirection
  - 4XX – client error
  - 5XX – server error
- Response code text phrase

Response status codes

In first line in server->client response message.

A few commonly occurring 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`

http message format: response

```
HTTP/1.0 200 OK
Date: Thu, 25 Aug 2001 12:00:15 GMT
Server: Apache/1.3.0 (Unix)
Last-Modified: Mon, 22 Aug 2001 …...
Content-Length: 6821
Content-Type: text/html

data data data data data ...
data data data data data ...
data data data data data ...
```

status line: (protocol status code status phrase)

header lines

data, e.g., requested html file

Try out HTTP (client side) for yourself

1. Telnet to your favorite Web server, e.g.:

   telnet www.utsc.utoronto.ca 80

   Open TCP connection to port 80 (default http server port) at www.utsc.utoronto.ca. Anything typed is sent to port 80 at www.utsc.utoronto.ca

2. Type in a GET HTTP request:

   GET /~rosselet/cscc09/index.html HTTP/1.0

   Type this at the prompt (followed by two carriage returns) to send this minimal GET request to the HTTP server

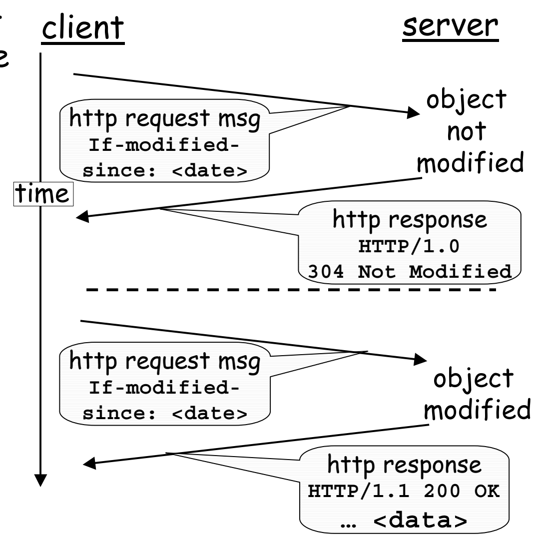
3. Look at response message sent by http server!

HTTP 1.0 other features

- POST
  - Client can send information to server
  - Forms, annotations
- If-modified-since request header
  - Client tells server it has data and asks server whether it has fresher version or client is up to date

User-server interaction: conditional GET

- Goal: don't send object if client has up-to-date copy (cached)
- client: 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



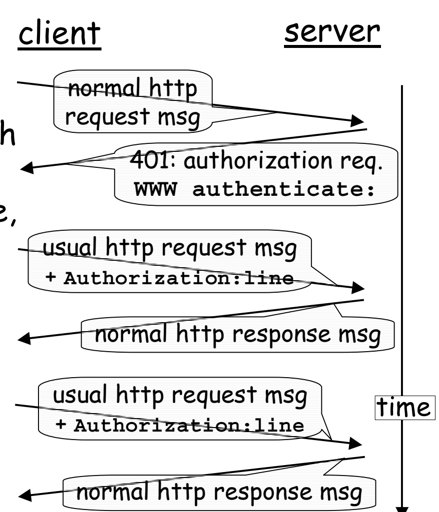
HTTP 1.0 authentication

- Basic authentication.
  - When challenged, client sends user id and password in clear to server
  - Not secure enough (snooping is easy) but useful for simple things

User-server interaction: authentication

Authentication goal: control access to server documents

- stateless: client must present authorization in each request
- authorization: typically name, password
  - **`authorization:`** header line in request
  - if no authorization presented, server refuses access, sends **`"WWW authenticate:"`** header line in response



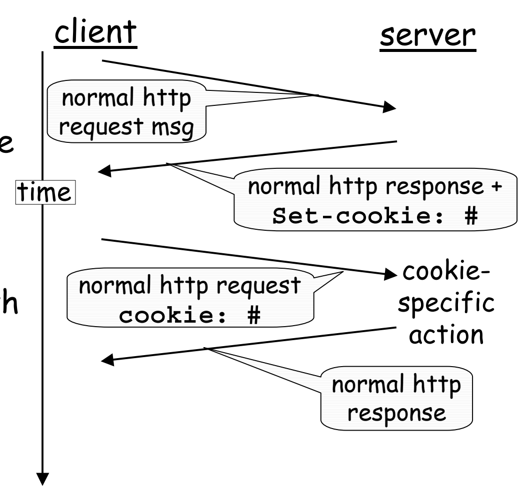
Browser caches name & password so that user does not have to repeatedly enter it.

Cookies

- Problem: HTTP is stateless
  - Server does not maintain status information across client requests
  - No way to correlate multiple request from same user
- Solution: store *cookie* on client side.
  - Small amount of information (typically server-generated user id)
  - Sent by client with each request
  - Updated by server with response

User-server interaction: cookies

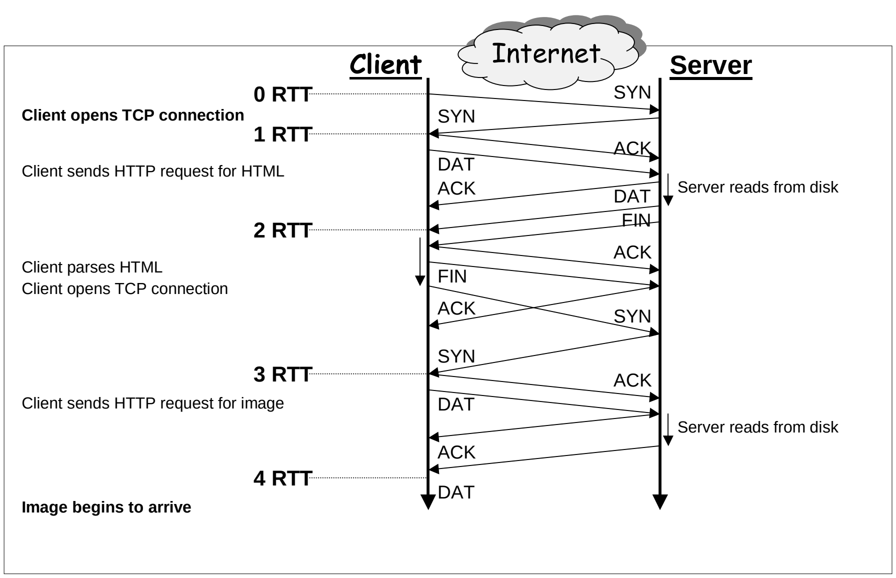
- server sends "cookie" to client in response msg
  `Set-cookie: 1678453`
- client presents cookie in later requests
  `cookie: 1678453`
- server matches presented-cookie with server-stored info
  - authentication
  - remembering user preferences, previous choices



HTTP 1.0: Problems

- Each request opens new connection
  - Starting up is slow (why?)
  - Takes several packets (why?)

Web Page with Single Image



More Problems

- Short transfers are hard on TCP
  - Stuck in "slow start" phase of TCP connection
  - Loss recovery is poor when windows are small
- Lots of extra connections
  - Increases server state/processing
- Server also forced to keep TIME_WAIT connection state
  - Why must server keep these?

Netscape Solution

- Use multiple concurrent connections to improve response time
  - Different parts of Web page arrive independently
  - Can grab more of the network bandwidth than other users
- Doesn't necessarily improve response time
  - TCP loss recovery ends up being timeout dominated because windows are small

HTTP 1.1: Persistent Connections

- Keeps connection open for a time after server response so that multiple requests can ride on single connection

  -> reduced connection setup overhead.

  GET index.html

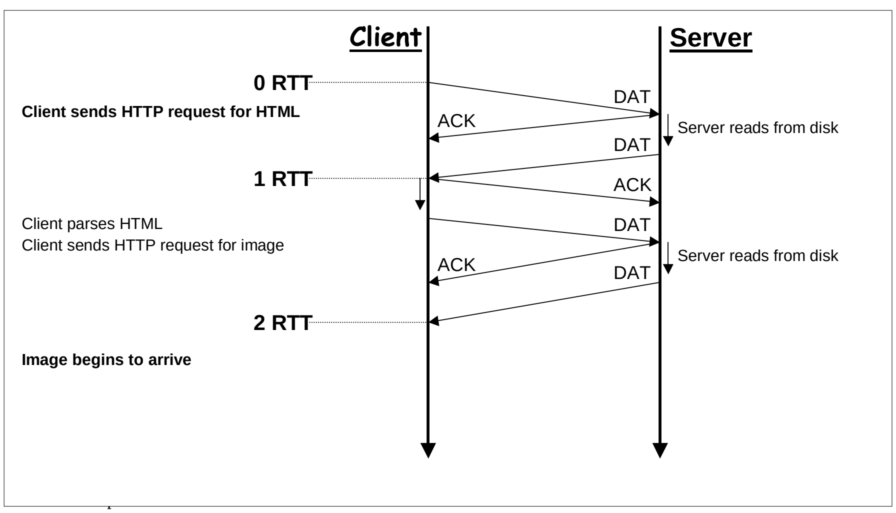
  Connection: keep-alive

  ... multiple HTTP requests ...

  Get banner.gif

  Connection: close

Persistent Connections



Connection length

- When does the data end?
  - Without persistent connections, when connection closes.
  - With persistent connections, reply header includes content length.

Non-persistent and persistent connections

Non-persistent

- HTTP/1.0
- server parses request, responds, and closes TCP connection
- 2 RTTs to fetch each object
- Each object transfer suffers from slow start

But most 1.0 browsers use parallel TCP connections.

Persistent

- default for HTTP/1.1
- on same TCP connection: server, parses request, responds, parses new request,..
- Client sends requests for all referenced objects as soon as it receives base HTML.
- Fewer RTTs and less slow start.

Persistent Connections

- Serialized requests do not improve response time.
- Pipelining requests.
  - Getall – request HTML document and all embeds
    - Requires server to parse HTML files
    - Doesn't consider client cached documents
  - Getlist – request a set of documents
    - Implemented as a simple set of GETs
- Prefetching
  - Must carefully balance impact of unused data transfers.

30 - http CSC309 45

Persistent Connection Performance

- Benefits greatest for small objects.
- Server resource utilization reduced due to fewer connection establishments and fewer active connections.
- TCP behavior improved.
  - Longer connections help adaptation to available bandwidth.
  - Larger congestion window improves loss recovery.

30 - http CSC309 46

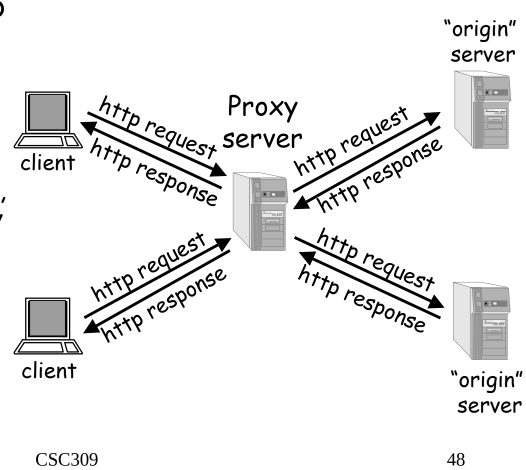
Caching

- Improve performance
  - Scalability
  - Response time
  - Load balancing
  - Availability
  - Saves network and server resources
- Proxy cache
  - Done at the client side

Web Caches (proxy server)

Goal: fill client request without going to origin server

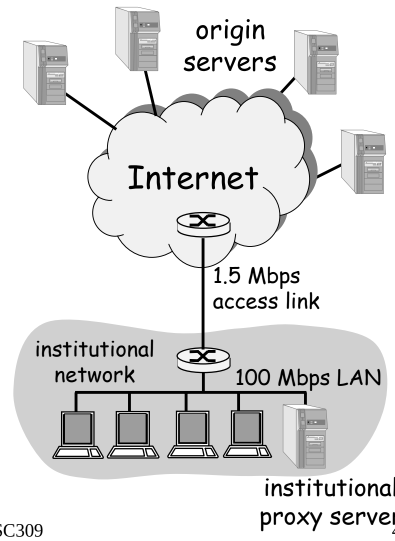
- user sets browser: Web accesses via web cache
- client sends all http requests to web cache
  - if object at web cache, web cache immediately returns object in http response
  - else requests object from origin server, then returns http response to client



Benefits of Web Caching

Assume: cache is "close" to client (e.g., in same network)

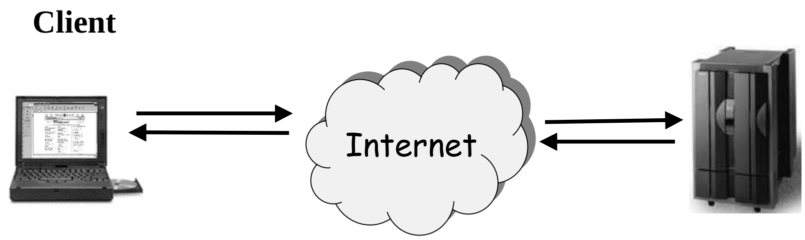
- smaller response time: cache "closer" to client
- decrease traffic to distant servers
  - link out of institutional/local ISP network often bottleneck



Caching architectures

- Proxy Caches
  - Serve a specific client population.
  - Can store recently accessed documents.
    - Lower latency for the end user.
    - Better use of wide area bandwidth by avoiding repeated transfers of recently-used information.
- Cooperative caching.
  - Multiple communicating caches.
  - … to increase hit rate and reduce access latency.
  - Approaches to cooperative caching
    - Hierarchical caches.
    - Non-hierarchical routable cache systems.

How do hosts find one another?



DNS: Domain Name System

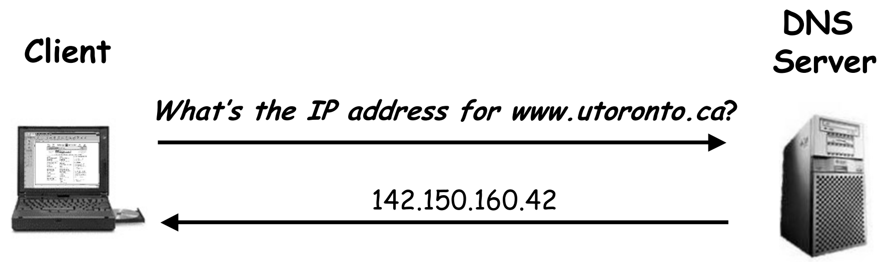
People: many identifiers:
- SSN, name, Passport #

Internet hosts, routers:
- IP address (32 bit) - used for addressing datagrams
- "name", e.g., www.scar.utoronto.ca - used by humans

Q: map between IP addresses and name ?

Domain Name System:
- *distributed database* implemented in hierarchy of many *name servers*
- *application-layer protocol* host, routers, name servers to communicate to *resolve* names (address/name translation)
  - note: core Internet function implemented as application-layer protocol
  - complexity at network's "edge"

Finding IP Address: Domain Name System (DNS)



Addressing

- Domain name (e.g. *www.utoronto.ca)*
  - Globally valid, human readable name.
- DNS translates name to IP address. (e.g. 142.150.160.42)
  - Globally valid, understood by all networks.
- Finally, we need local area net address.
  - E.g., Ethernet (*08-00-4a-22-1b-98)*
  - Globally unique, but used only within a particular local area network (LAN)
  - … more on this later …

DNS name servers

Why not centralize DNS?

- single point of failure
- traffic volume
- distant centralized database
- maintenance

doesn't *scale!*

- no server has all name-to-IP address mappings

local name servers:

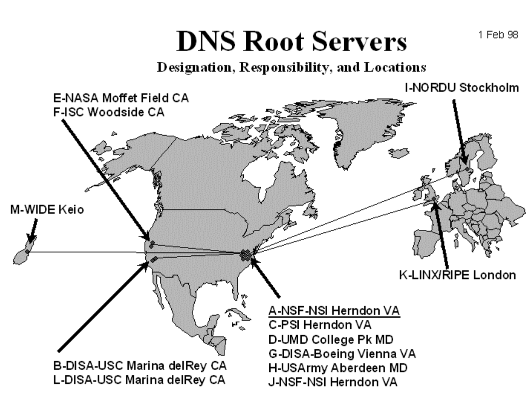
- each ISP, company has *local (default) name server*
- host DNS query first goes to local name server

authoritative name server:

- for a host: stores that host's IP address, name
- can perform name/address translation for that host's name

DNS: Root name servers

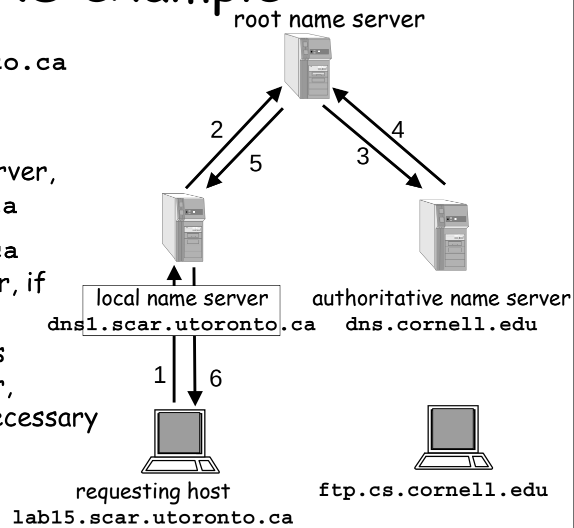
- contacted by local name server that can not resolve name
- root name server:
  - contacts authoritative name server if name mapping not known
  - gets mapping
  - returns mapping to local name server
- ~ dozen root name servers worldwide



Simple DNS example

Host **lab15.scar.utoronto.ca** wants IP address of **ftp.cs.cornell.edu**

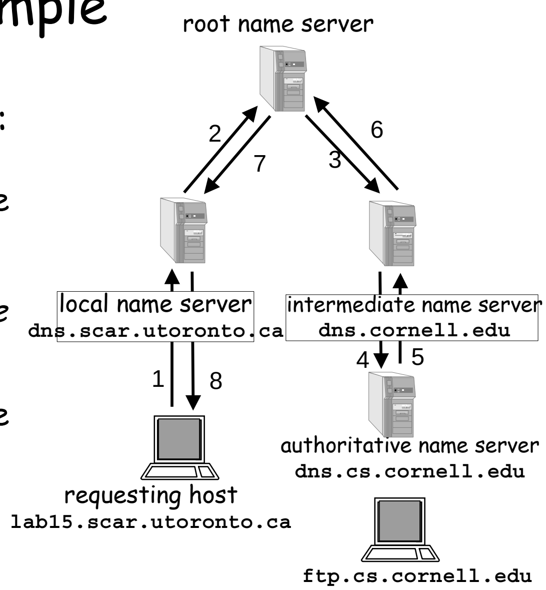
1. Contacts its local DNS server, **dns1.scar.utoronto.ca**
2. **dns1.scar.utoronto.ca** contacts root name server, if necessary
3. root name server contacts authoritative name server, **dns.cornell.edu**, if necessary



DNS example

Root name server:

- may not know authoritative name server
- may know *intermediate name server:* who to contact to find authoritative name server



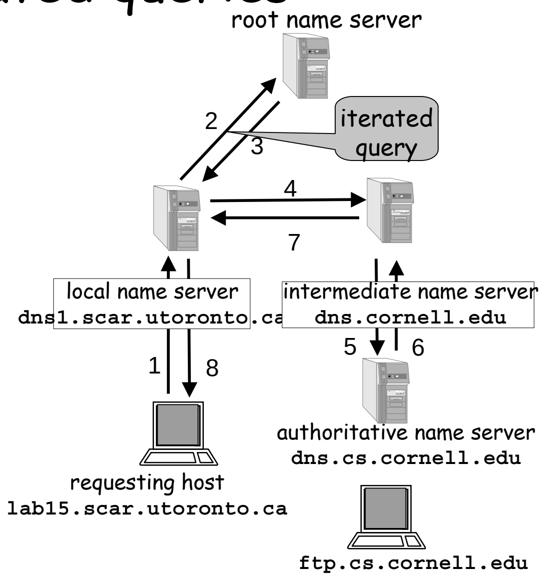
DNS: iterated queries

recursive query:

- puts burden of name resolution on contacted name server
- heavy load?

iterated query:

- contacted server replies with name of server to contact
- "I don't know the answer, ask this server"



DNS: caching and updating records

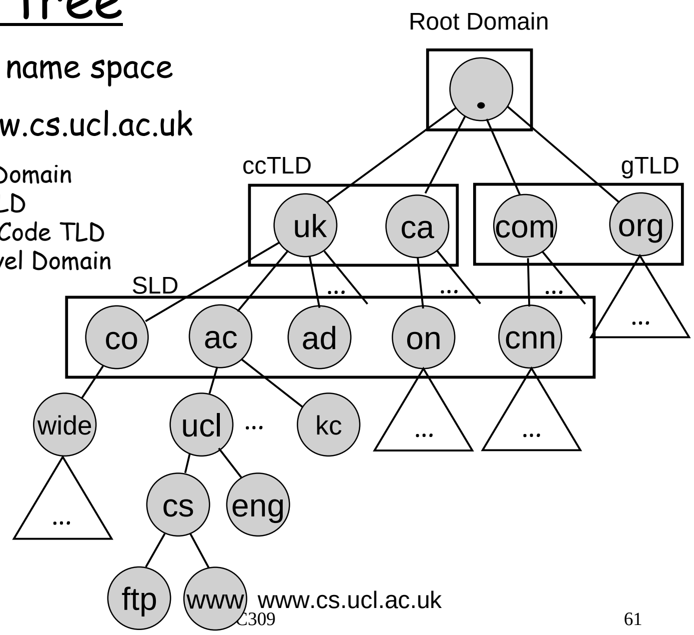
- once (any) name server learns mapping, it *caches* mapping
  - cache entries timeout (disappear) after some time
- update/notify mechanisms under design by IETF
  - RFC 2136
  - http://www.ietf.org/html.charters/dnsind-charter.html

Domain tree

Hierarchical name space

Example: www.cs.ucl.ac.uk

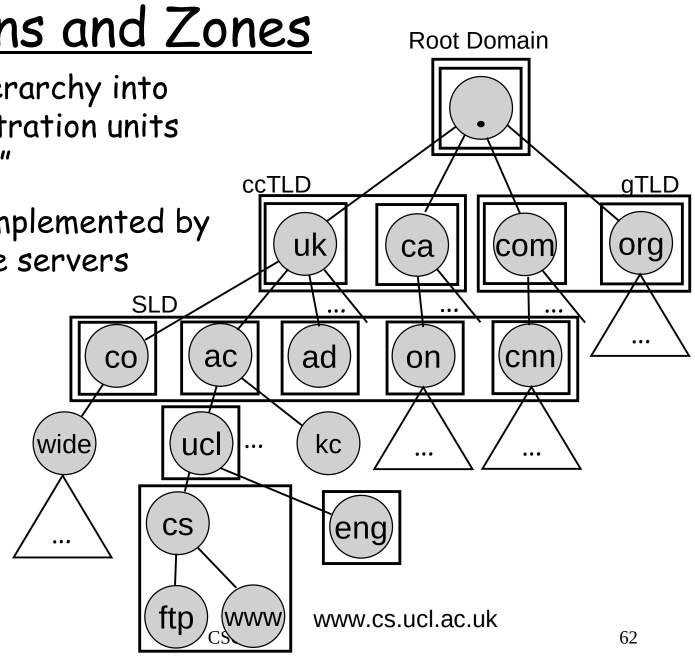
TLD: Top Level Domain
gTLD: generic TLD
ccTLD: Country Code TLD
SLD: Second Level Domain



Domains and Zones

•Partition hierarchy into data administration units called "zones"

•Each zone implemented by a set of name servers



DNS records

DNS: distributed db storing resource records (RR)

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

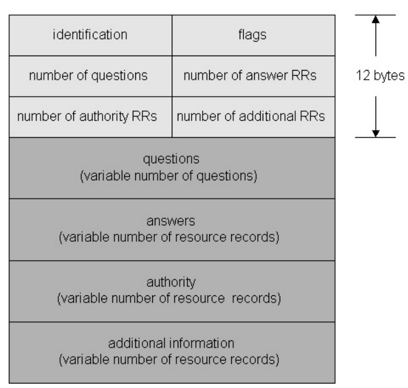
- Type=A
  - **name** is hostname
  - **value** is IP
- Type=NS
  - **name** is domain (e.g. foo.com)
  - **value** is IP address of authoritative name server for this domain address
- Type=CNAME
  - **name** is an alias name for some "cannonical" (the real) name
  - **value** is canonical name
- Type=MX
  - **value** is hostname of mailserver associated with **name**

DNS protocol, messages

DNS protocol : *query* and *reply* messages, both with same *message format*

msg header

- identification: 16 bit # for query, reply to query uses same #
- flags:
  - query or reply
  - recursion desired
  - recursion available
  - reply is authoritative



DNS protocol, messages