Application Layer Protocols -1

A/PROF. DUY NGO

Learning Objectives

2.1 Principles of network applications

2.2 Web and HTTP

2.4 DNS

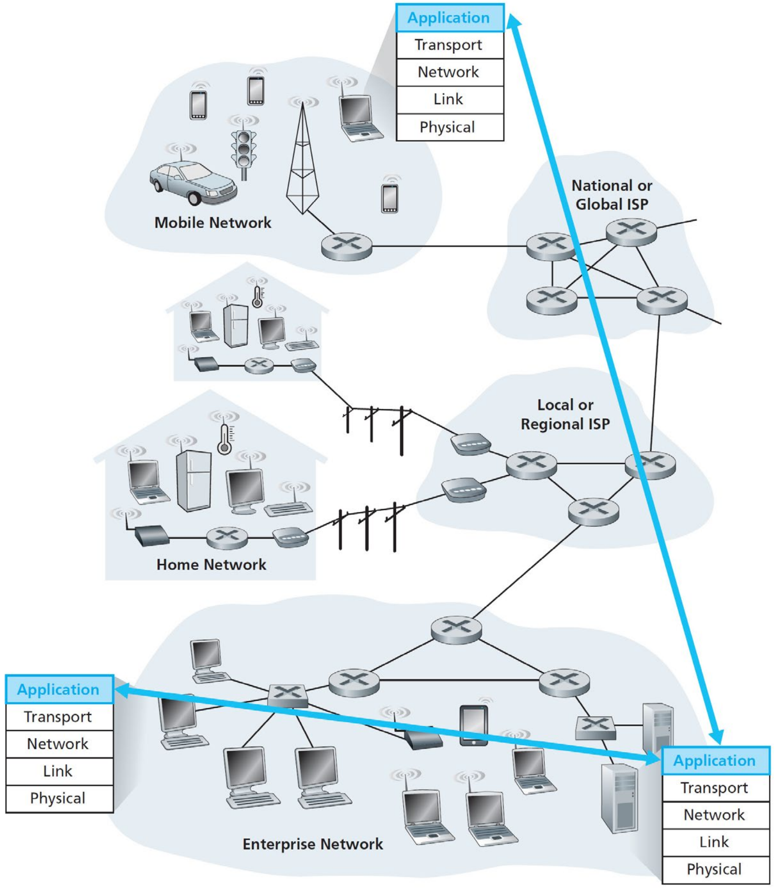
Creating a Network App

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



Application Architectures

Possible structure of applications:

- client-server
- peer-to-peer (P2P)

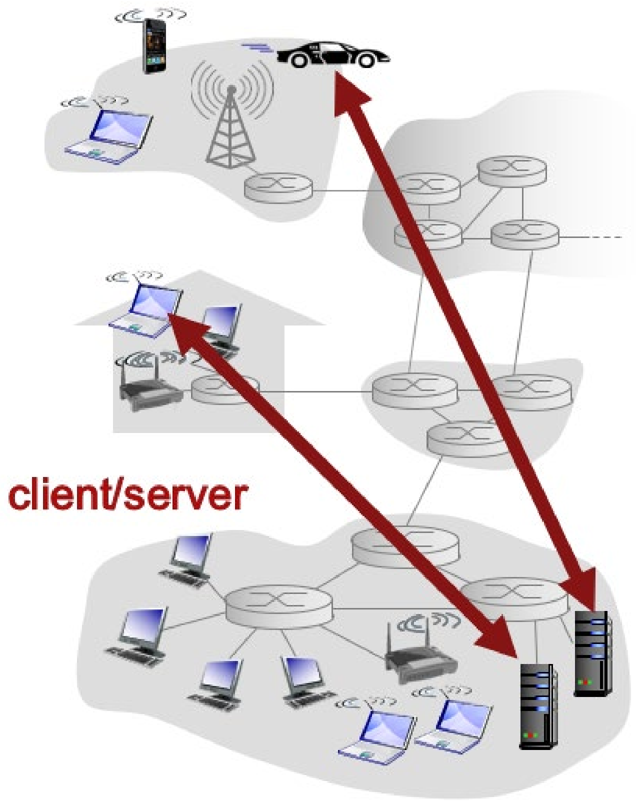
Client-Server Architecture

server:

- always-on host
- permanent IP address
- data centers for scaling

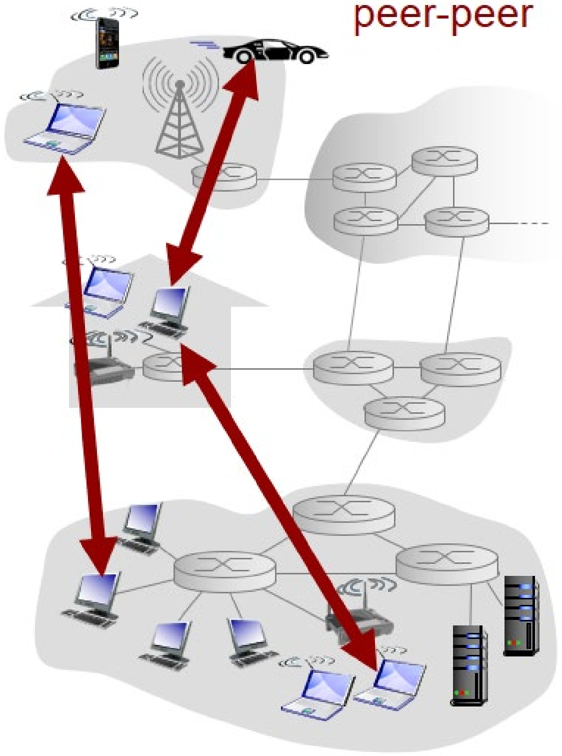
clients:

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



P2P (Peer to Peer) Architecture

- **Minimum** reliance on dedicated servers
- arbitrary end systems directly communicate
- peers request service from other peers, provide service in return to other peers
  - **self scalability** – new peers bring new service capacity, as well as new service demands
- peers are intermittently connected and change IP addresses
  - complex management



Processes Communicating

process*:* program running within a host

- within same host, two processes communicate using **inter-process communication** (defined by OS: Operating System)
- processes in different hosts communicate by exchanging **messages**

clients, servers

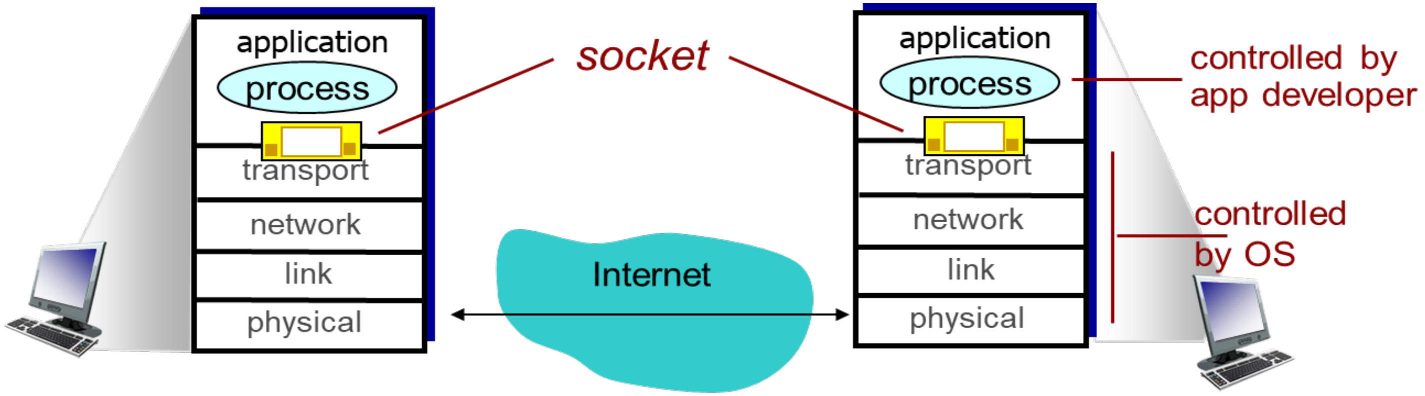
client process: process that initiates communication

server process: process that waits to be contacted

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

Sockets

- process sends/receives messages to/from its **socket**
- socket analogous to door
  - sending process sends message to out door
  - sending process relies on transport infrastructure on other side of door to deliver message to socket at receiving 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
  - 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

open protocols:

- defined in RFCs
- allows for interoperability
- e.g., HTTP, SMTP

proprietary protocols:

- e.g., Skype

What Transport Service Does An App Need?

data integrity

- some apps (e.g., file transfer, web transactions) require 100% reliable data transfer
- other apps (e.g., audio) can tolerate some loss

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: 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
text messaging	no loss	elastic	yes and no

Internet Transport Protocols Services

TCP service:

- **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 guarantee, security
- **connection-oriented:** setup required between client and server processes

UDP service:

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

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 (e.g., YouTube), RTP [RFC 1889]	TCP or UDP
Internet telephony	SIP, RTP, proprietary (e.g., Skype)	TCP or UDP

Securing TCP

TCP & UDP

- no encryption
- cleartext passwds sent into socket traverse Internet in cleartext

SSL

- provides encrypted TCP connection
- data integrity
- end-point authentication

SSL is at app layer

- apps use SSL libraries, that "talk" to TC P

SSL socket API

- cleartext passwords sent into socket traverse Internet encrypted
- see Chapter 8

Web and HTTP

First, a review…

- **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,** e.g.,

$$\underbrace{\texttt{www.someschool.edu}}_{\text{host name}}\underbrace{\texttt{/someDept/pic.gif}}_{\text{path name}}$$

HTTP Overview (1 of 2)

HTTP: hypertext transfer protocol

Web's application layer protocol

client/server model

- **client:** browser that requests, receives, (using HTTP protocol) and "displays" Web objects
- **server:** Web server sends (using HTTP protocol) objects in response to requests

HTTP Overview (2 of 2)

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

non-persistent HTTP

- at most one object sent over TCP connection
  - connection then closed
- downloading multiple objects required multiple connections

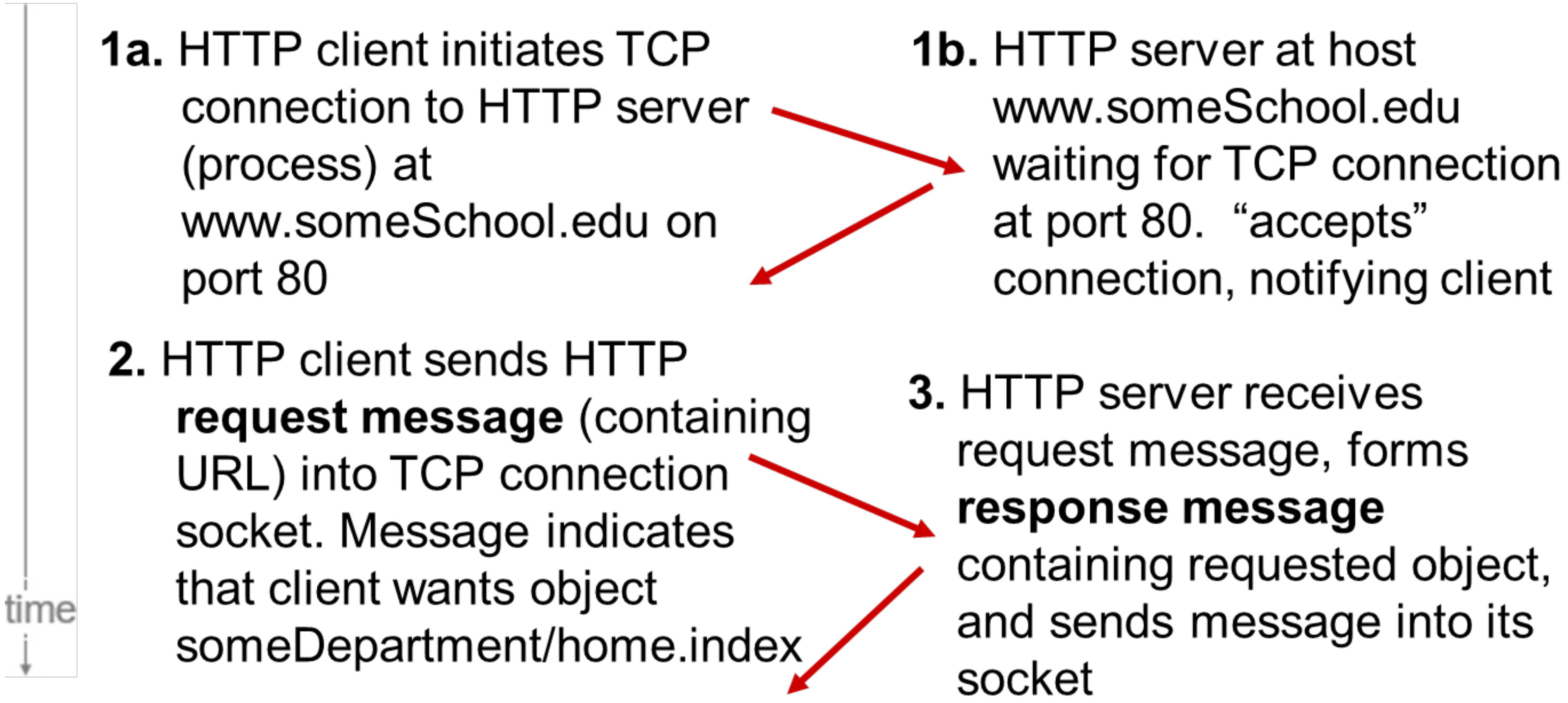
persistent HTTP

- multiple objects can be sent over single TCP connection between client, server

Non-Persistent HTTP (1 of 2)

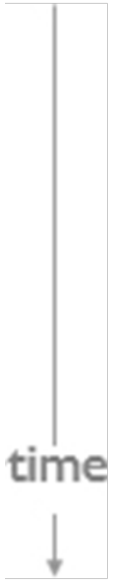
suppose user enters URL:

www.someSchool.edu/someDepartment/home.index



Non-Persistent HTTP (2 of 2)

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

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

Non-Persistent HTTP: Response Time

RTT (definition): time for a small packet to travel from client to server and back

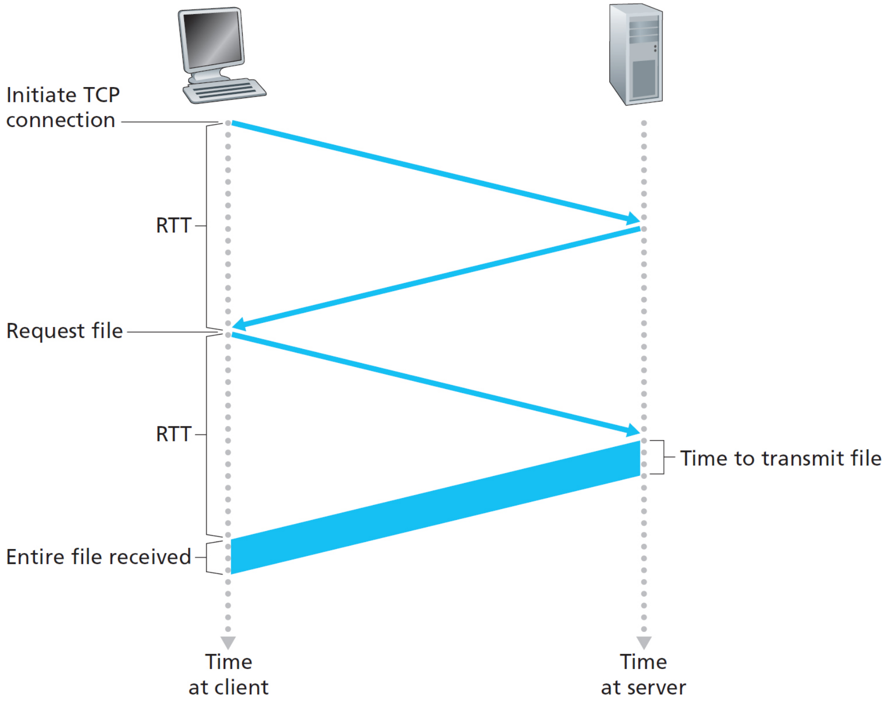
HTTP response time:

one RTT to initiate TCP connection

one RTT for HTTP request and first few bytes of HTTP response to return

file transmission time

non-persistent HTTP response time = 2RTT+ file transmission time



Persistent HTTP

non-persistent HTTP issues:

- requires 2 RTT s 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)

carriage return character

line-feed character

request line (GET, POST, HEAD commands)

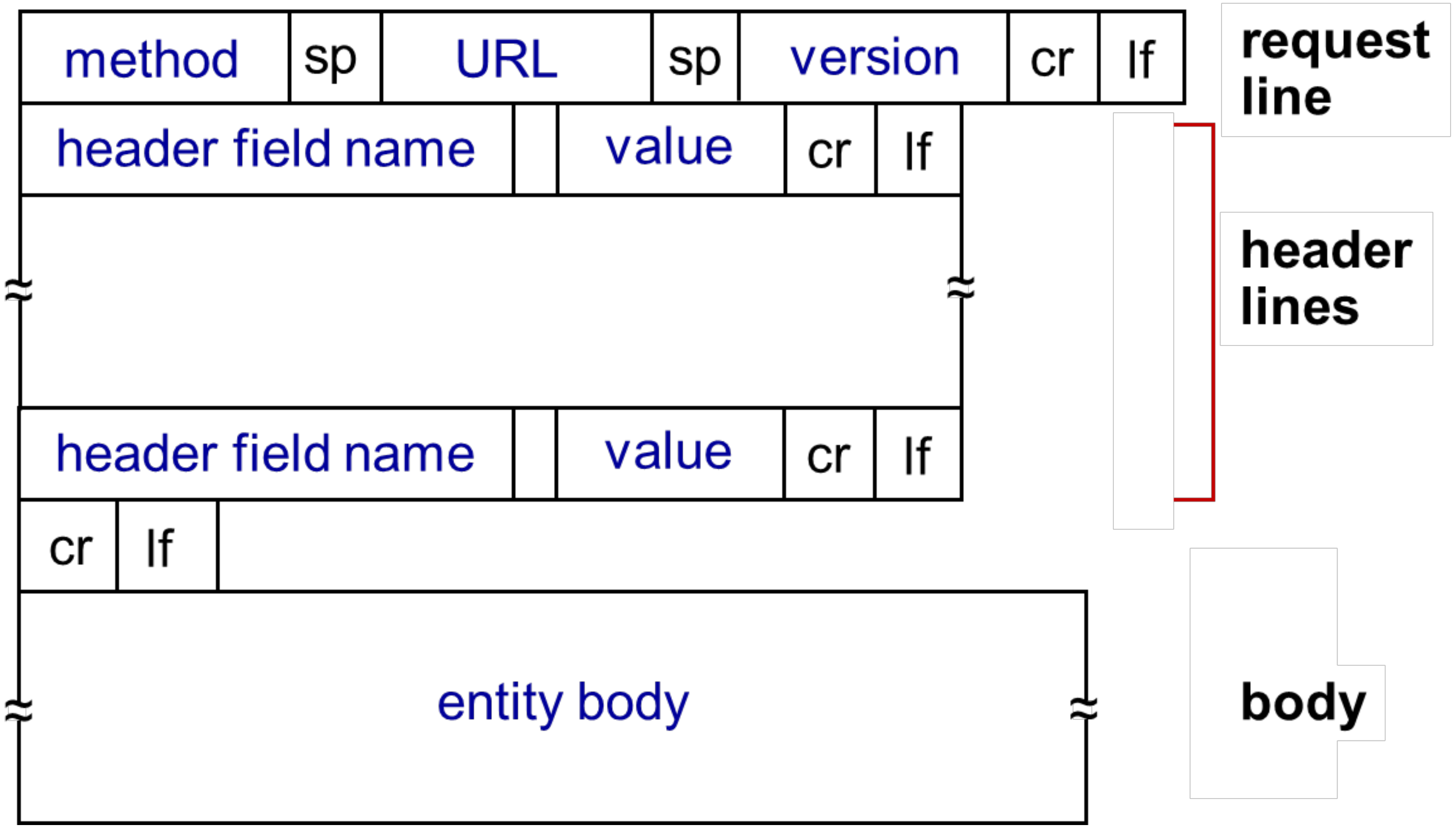
header lines

carriage return, line feed at start of line indicates end of header lines

```
GET /index.html HTTP/1.1\r\n
Host: www-net.cs.umass.edu\r\n
User-Agent: Firefox/3.6.10\r\n
Accept: text/html,application/xhtml+xml\r\n
Accept-Language: en-us,en;q=0.5\r\n
Accept-Encoding: gzip,deflate\r\n
Accept-Charset: ISO-8859-1,utf-8;q=0.7\r\n
Keep-Alive: 115\r\n
Connection: keep-alive\r\n
\r\n
```

* Check out the online interactive exercises for more examples: http://gaia.cs.umass.edu/kurose_ross/interactive/

HTTP Request Message: General Format



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.yahoo.com - used by humans

Q: how to map between IP address and name, and vice versa?

Domain Name System:

- **distributed database** implemented in hierarchy of many **name servers**
- **application-layer protocol:** hosts, name servers communicate to **resolve** names (address/name translation)
  - note: core Internet function, implemented as application-layer protocol
  - complexity at network’s “edge”

DNS: Services, Structure

DNS services

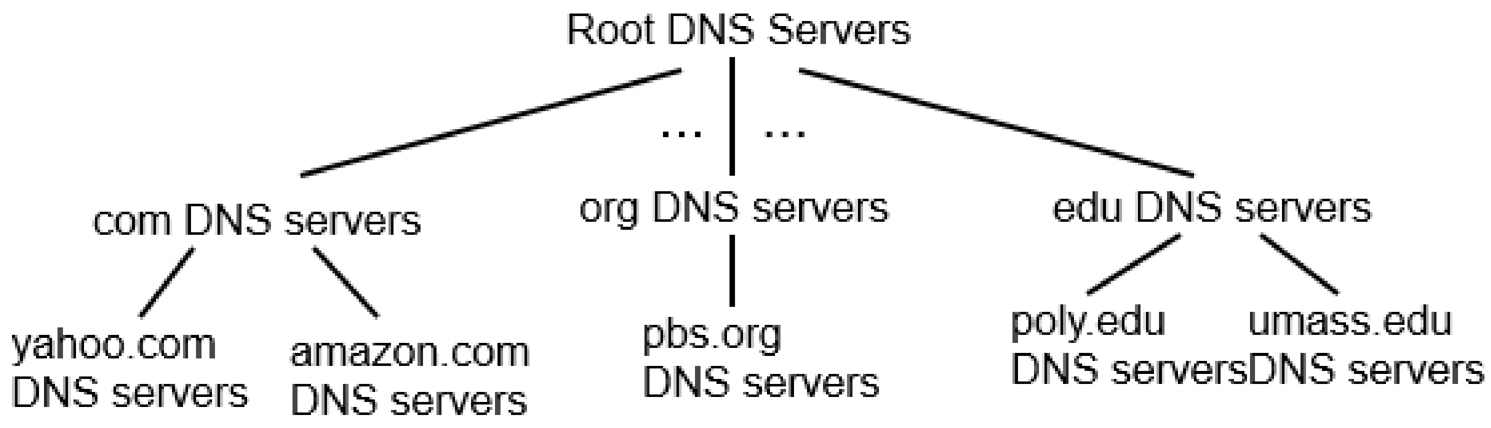
- hostname to IP address translation
- host aliasing
  - canonical, alias names
- mail server aliasing
- load distribution
  - replicated Web servers: many IP addresses correspond to one name

why not centralize DNS?

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

A: **doesn‘t scale!**

DNS: A Distributed, Hierarchical Database



client wants IP for www.amazon.com;

1st approximation:

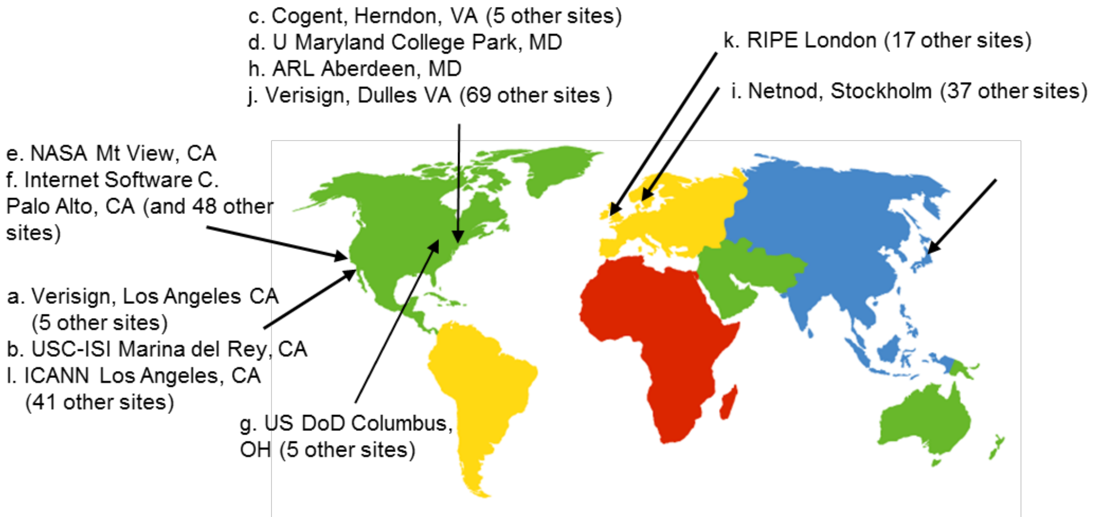
- client queries root server to find com DNS server
- client queries .com DNS server to get amazon.com DNS server
- client queries amazon.com DNS server to get IP address for www.amazon.com

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

13 logical root name "servers" worldwide

- **each "server" replicated many times**



TLD, Authoritative Servers

top-level domain (TLD) servers:

- responsible for com, org, net, edu, aero, jobs, museums, and all top-level country domains, e.g.: uk, fr, ca, jp
- Network Solutions maintains servers for .com TLD
- Educause for .edu TLD

authoritative DNS servers:

- Organization’s own DNS server(s), providing authoritative hostname to IP mappings for organization’s named hosts
- can be maintained by organization or service provider

Local DNS Name Server

does not strictly belong to hierarchy

each ISP (residential ISP, company, university) has one

- also called “default name server”

when host makes DNS query, query is sent to its local DNS server

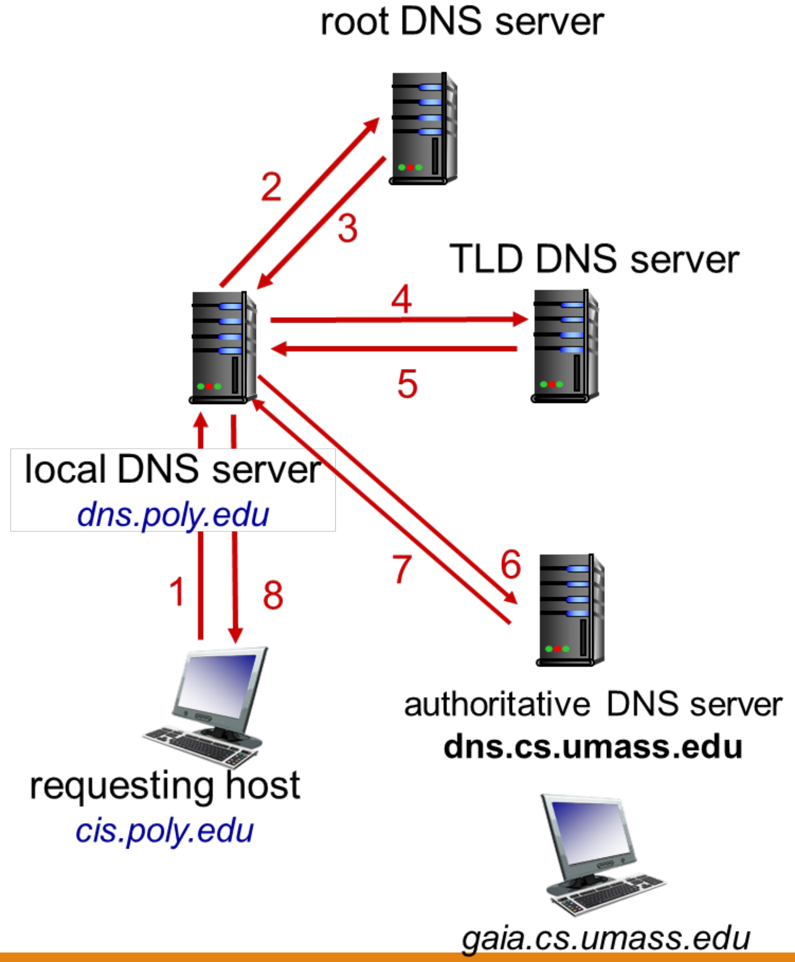
- has local cache of recent name-to-address translation pairs (but may be out of date!)
- acts as proxy, forwards query into hierarchy

DNS Name Resolution Example (1 of 2)

- host at cis.poly.edu wants IP address for gaia.cs.umass.edu

iterated query:

- contacted server replies with name of server to contact
- “I don’t know this name, but ask this server”

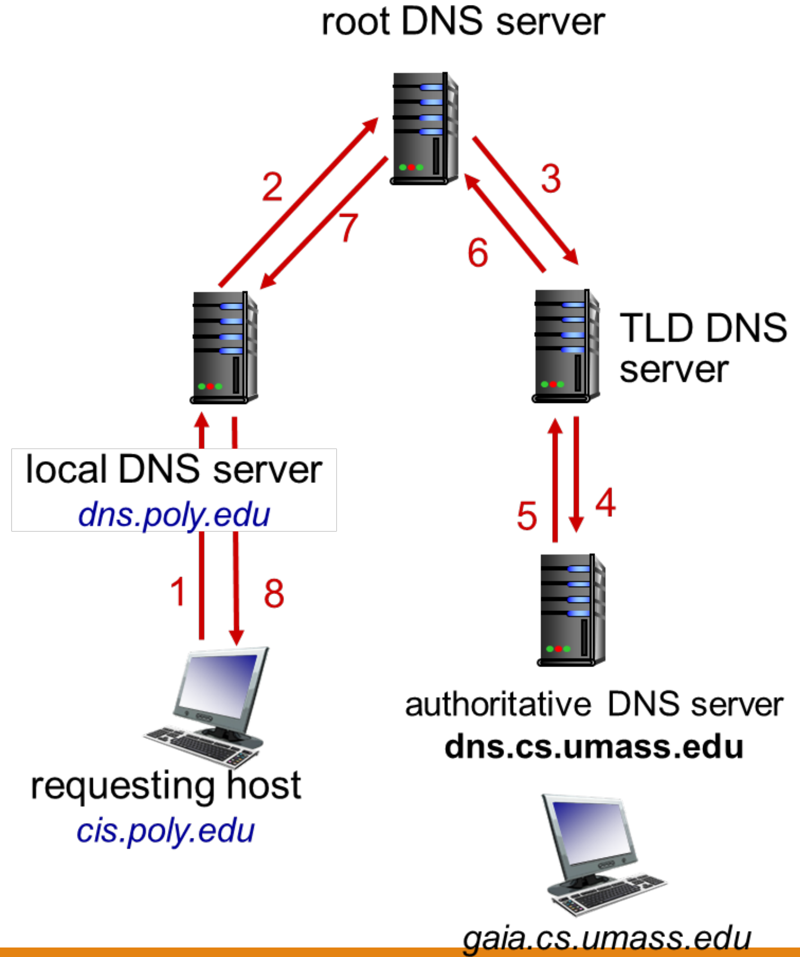


DNS Name Resolution Example (2 of 2)

recursive query:

puts burden of name resolution on contacted name server

heavy load at upper levels of hierarchy?



DNS: Caching, Updating Records

once (any) name server learns mapping, it **caches** mapping

- cache entries timeout (disappear) after some time (TTL)
- TLD servers typically cached in local name servers
  - thus root name servers not often visited

cached entries may be **out-of-date** (best effort name-to-address translation!)

- if name host changes IP address, may not be known Internet-wide until all TTL s expire

update/notify mechanisms proposed IETF standard

- RFC 2136

DNS Records

DNS: distributed database storing resource records **(RR)**

RR format: `(name, value, type, 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

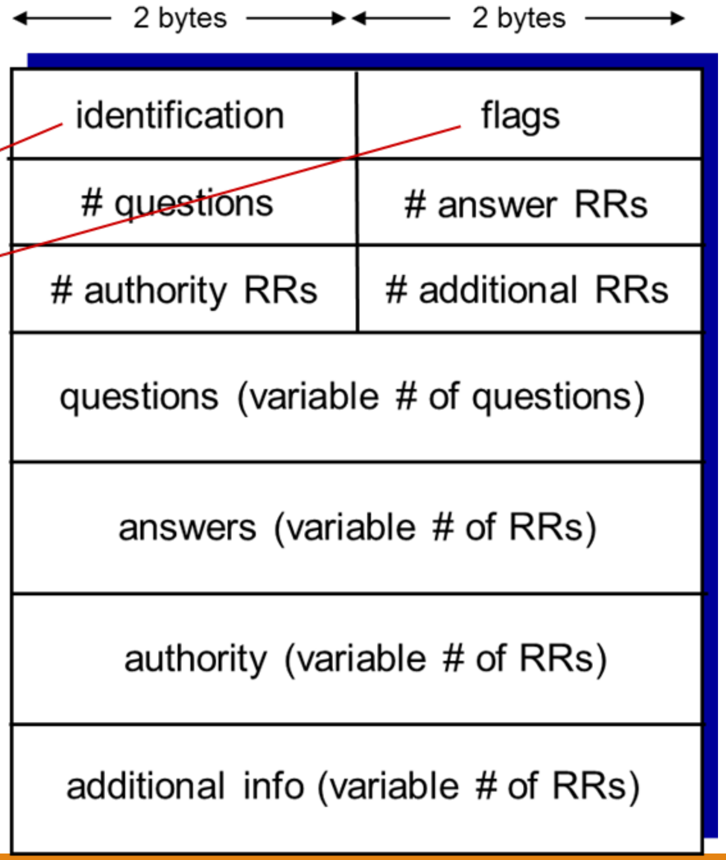
- `value` is name of mailserver associated with `name`

DNS Protocol, Messages (1 of 2)

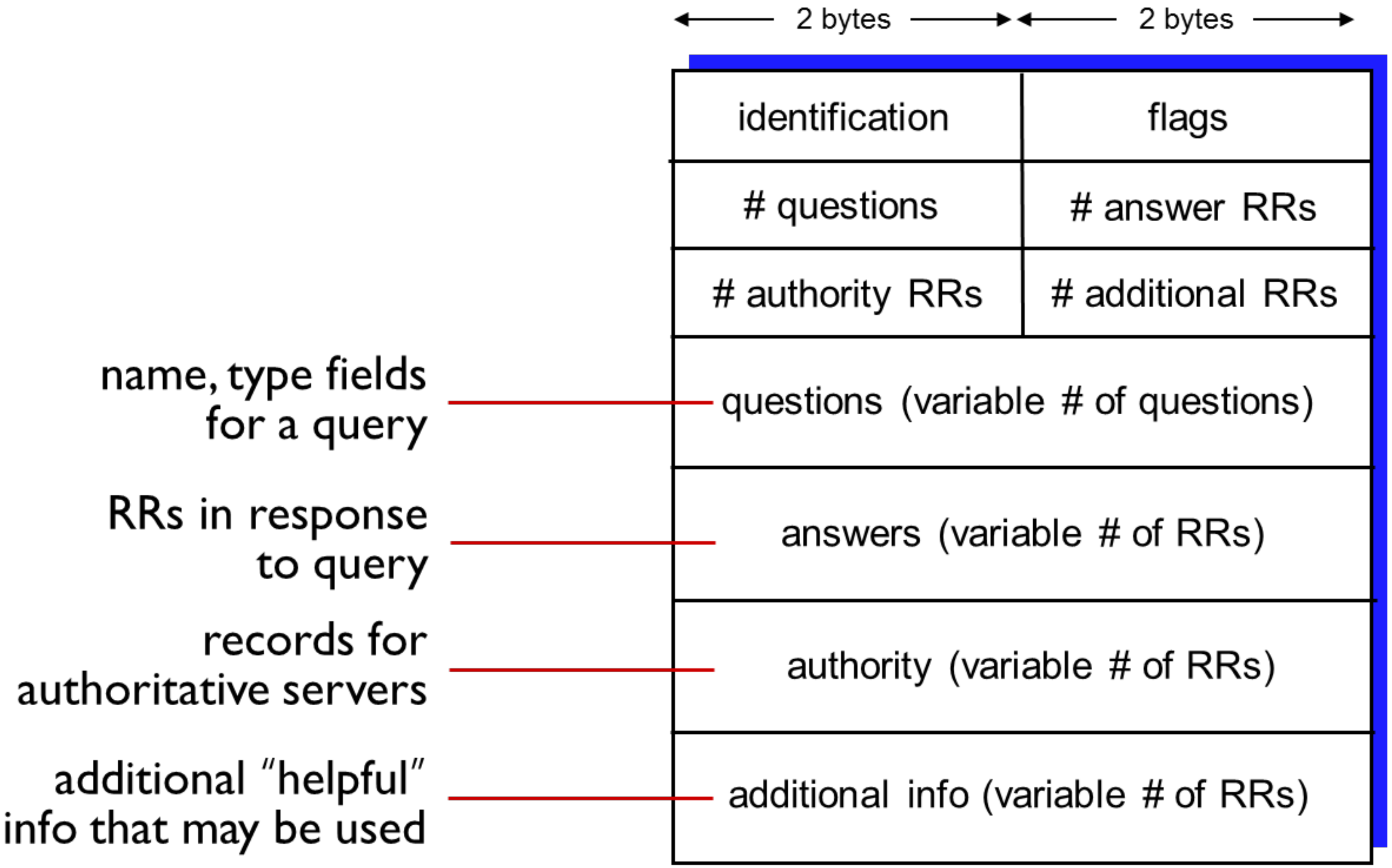
- **query** and **reply** messages, both with same **message format**

message 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 (2 of 2)



Attacking DNS

DDoS attacks

- bombard root servers with traffic
  - not successful to date
  - traffic filtering
  - local DNS servers cache IPs of TLD servers, allowing root server bypass
- bombard TLD servers
  - potentially more dangerous

redirect attacks

- man-in-middle
  - Intercept queries
- DNS poisoning
  - Send bogus replies to DNS server, which caches

exploit DNS for DDoS

- send queries with spoofed source address: target IP
- requires amplification

