Application architectures

possible structure of applications:

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

Problems in P2P:

1. ISP Friendly. Most residential ISPs (including DSL and cable ISPs) have been

dimensioned for “asymmetrical” bandwidth usage, that is, for much more downstream than upstream traffic.

2) Security. Because of their highly distributed and open nature, P2P applications

can be a challenge to secure

3) Incentives. The success of future P2P applications also depends on convincing

users to volunteer bandwidth, storage, and computation resources to the applications

Addressing Process

To get receiving process, we must have 2 things

1. Address of end host (IP) 2) port number (Many processes can run on end host)

Eg . Web server has port 80 , Mail (SMTP) has Port 25

Transport Services Available : Basis to choose

1. Reliable Data Transfer

Guaranteed Data delivery service

2) ThroughPut (rate at which the sending process can deliver bits to receiving process)

Guaranteed Available throughput

Apps that have throughput requirement are called bandwidth-sensitive

3) Timing

No Delay

4) Security

Encrypt, data integrity, end-point authentication

TCP : Reliable, Flow Control, Congestion Control, Connection oriented

UDP : Unreliable, connectionless, no guarantee or delivery/order, no congestion control

**Transport cannot guarantee throughput or timing.**

Application Layer Protocols

It defines :

* The types of messages exchanged, for example, request messages and response messages
* The syntax of the various message types, such as the fields in the message and how the fields are delineated
* The semantics of the fields, that is, the meaning of the information in the fields
* Rules for determining when and how a process sends messages and responds to messages

WEB and HTTP

Overview of HTTP

* Implemented in two programs : Client (Browser that requests) and Server (Stores and sends data when requested)
* Uses TCP as its Transport Protocol
* Takes messages from socket interface and throws responses in socket
* Server send info to client without storing information about client : STATELESS

Persistent and Nonpersistent

When this client-server interaction is taking place over TCP, the application developer needs to make an important decision––should each request/response pair be sent over a separate TCP connection, or should all of the requests and their corresponding responses be sent over the same TCP connection.

Persistent : Same Connection

Nonpersistent : New connection for each message

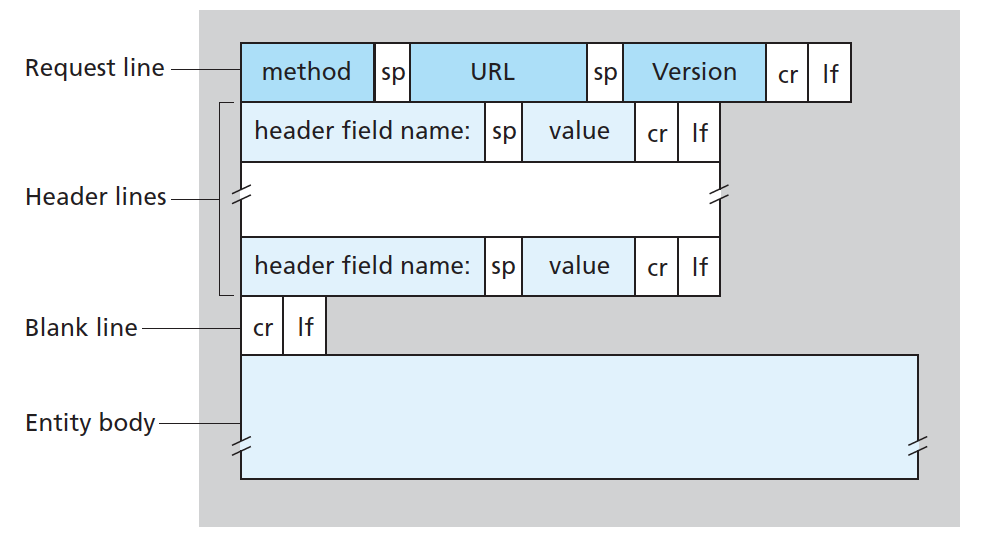
NONPERSISTENT

* **At Most One** object sent over the conn, then the conn is closed
* Each object opens a new conn
* If web server has 1 base file and 10 images : 11 connections made
* Initiate TCP with server -> Send request for path name -> Server send info to client -> First conn closed -> Client gets HTTP mesg (Finds 10 image references) -> New Conn for each
* Each TCP Conn transports EXACTLY 1 request and 1 response message
* Most browsers can open 5-10 Parallel TCP Conns
* RESPONSE TIME
  + Round Trip Time : which is the time it takes for a small packet to travel from client to server and then back to the client.
  + The RTT includes packet-propagation delays, packet queuing delays in intermediate routers and switches, and packet-processing delays.
  + Total Response time of one TCP is 2 RTT + Transmission time at server of HTTP
* Problems
  + New Conn for each request
  + Each conn requires buffers, variables maintained on both client and server
  + Each object takes Total Response time

PERSISTENT

* The server leaves the TCP connection open after sending a response
* Pipelining : These requests for objects can be made back-to-back, without waiting for replies to pending requests
* Default mode of TCP using pipelining

HTTP REQUEST MESSAGE ( Ascii Human Readable)



* Request Line : Has Three Fields
  + Method : GET,POST,HEAD,PUT,DELETE. (When request: GET Used)
  + URL : Any URL
  + Version : Eg HTTP/1.1
* HOST Header Line : Host on which requested object resides (Used in Caching)
* CONNECTION Header Line : What to do with conn, eg : Close
* USER-AGENT : Type of Browser, eg Mozilla
* Accept Language : eng, fr
* Entity : Empty in Get, Used in POST

GET : Puts everything in URL

POST : User inputs a form/search queries : Uses Entity to store such things

HEAD : Similar to get , but leaves out the required object (Used in debuggin)

PUT : Uploading objects to servers

DEL : delete an object

SAMPLE :

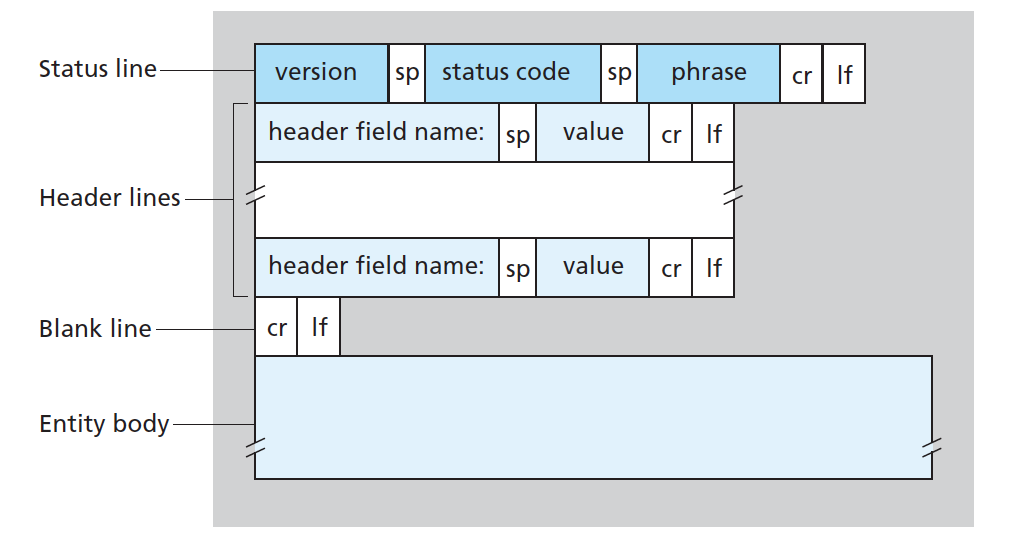
GET /somedir/page.html HTTP/1.1

Host: www.someschool.edu

Connection: close

User-agent: Mozilla/5.0

Accept-language: fr

HTTP RESPONSE MESSAGE

3 Parts : Status Line, **6** Header lines, Entity (Contains requested Data)

* STATUS LINE : 3 fields
  + Version
  + Response Code : 200, 301, 404, 400, 505
  + Response : OK, MOVED , NOT FOUND, BAD REQ
* HEADERS :
  + Connection : Close/Open
  + Date : Time when response was created and sent by server
  + Server : What type of server generated the mesg, Eg , APACHE
  + LAST MODIFIED : When the object being sent was last modified
  + Content Length : No. of bytes of object being sent
  + Content Type : text, jpeg …

SAMPLE

HTTP/1.1 200 OK

Connection: close

Date: Tue, 09 Aug 2011 15:44:04 GMT

Server: Apache/2.2.3 (CentOS)

Last-Modified: Tue, 09 Aug 2011 15:11:03 GMT

Content-Length: 6821

Content-Type: text/html

COOKIES : Keep Track of users

(1) a cookie header line in the HTTP response message;

(2) a cookie header line in the HTTP request message;

(3) a cookie file kept on the user’s end system and managed by the user’s browser;

(4) a back-end database at the Web site

CACHING

satisfy client request without involving origin server

GOOD WHY : 1) Reduces Response Time for a client request 2) Reduces Traffic on insti Link

Uses CONDITIONAL GET

Verify that a cached copy is up-to-date

**FTP**

Transfer to and from a remote host

PORT : 21

Uses 2 Parallel TCP connection : 1) Control Conn 2) Data Conn

Control Conn : Sending Control information : ID, Pwd, change directory, put and get

Data Conn : Used to actually send a file

IMP : 1 Control Conn, but a new Data Conn for each Request (Data is Non-Persistent)

Maintains State : users current direc/direc tree

COMMANDS and REPLIES

* 7 Bit ASCII : 4 uppercase ASCII with some options
* Common Commands
  + USER username
  + PASS password
  + LIST
  + RETR filename
  + STOR filename
* Replies
  + Resonse Codes

**ELECTRONIC MAIL**

3 major components :

1. User Agents 2) Mail Servers 3) Simple Mail Transfer Protocol

SMTP

* Main App-Layer protocol for email
* Uses TCP
* Two side : Client and Server : Run on every mail server
* Older than HTTP
* BAD : Restricts body of all messages to be 7 bit ASCII
* IMP : Does not use intermediate mail server to transfer mails
  + Direct TCP Conn between Sender and Receiver
* PORT : 25
* PERSISTENT ( One mail server can send several mails to another in 1 conn)
* SMTP is a PUSH protocol (The sender pushes file to the receiver while HTTP is PULL)
* Every character must be 7 bit ASCII (If not then encode in 7 bit and then send, while HTTP has no such restrictions)
* Different objects (text and images) are all sent in the same Message (While in HTTP they are sent in different mesages)

MAIL ACCESS PROTOCOL

* Required because once the message is in the receivers mail server, SMTP cant access it itself, this is because it is a PUSH protocol and not a PULL one (LOL)
* Special Mail Protocol to PULL Mail from mail server
  + POP3 (Post Office Protocol version 3)
  + Internet Mail Access Protocol
  + HTTP
* SMTP Tranfers from senders user agent to receivers mail server, these assist in getting mail from mail server into receivers user agent

POP3

* Begins when user agent opens TCP on mail server
* PORT 110
* POP3 can do just **2 responses : +OK and -ERR**
* 3 phases : Authorization, Transaction , Update
  + Authorization
    - User Agent sends username and password commands
    - +OK is correct else -ERR
  + Transaction
    - Retrieves messages, mark mesgs for deletion (unmark them), mail stats
    - Uses LIST, RETR and DELE commands of FTP
  + Updation
    - QUIT command
    - Ends sessions : IMP : Now all the marked mails are deleted from server
* Uses : “Download and Delete” or :Download and Keep” during transaction
  + DnD is bad since once done, mesg is deleted
* Saves some STATE in one session but not across sessions

IMAP

* Associates each message with a folder
* Maintains sessions across IMAP sessions
* User can create new folders, move mesgs blahblah
* Can be used to obtain only parts of the message : Just header …. Useful in low bandwidth connection

**Domain Name System**

Directory service that translates hostnames to IP’s

1. Distributed Database implemented in a hierarchy of DNS servers
2. App Layer Protocol that allows hosts to query databases

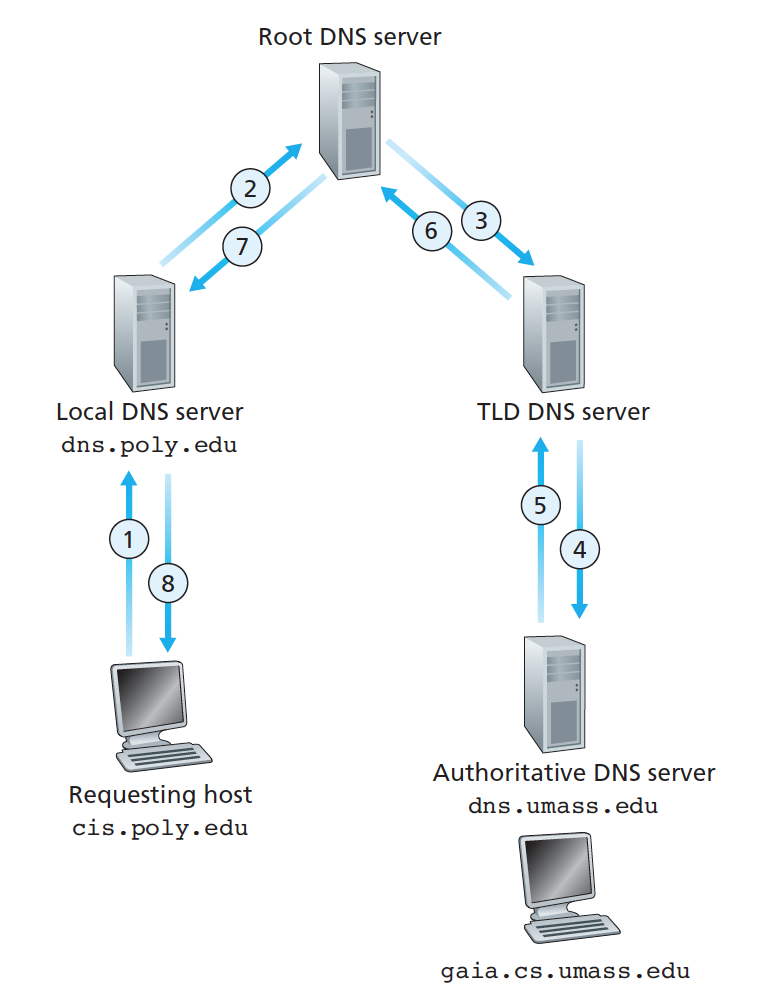
Use Berkeley Internet Name domain

**UDP and Port 53**

Other Jobs : 1) Host Aliasing 2) Mail Server Aliasing 3) Load Distribution

Distributed Database in three levels

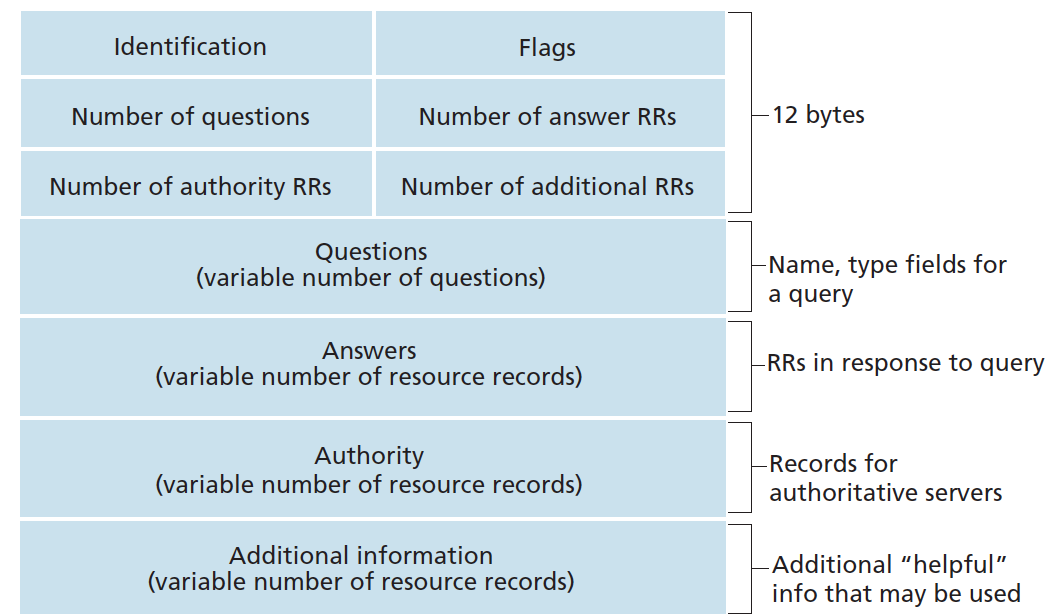
* Root DNS
  + 1st Class : Highest level
* Top Level Domain(TLD) server
  + Domains like .com, .edu, .org and country domains
* Authoritative
  + Inside an organisation
* Local
  + Further inside an organisation

 8 DNS Messages for a single query

DNS RECORDS

* 4 tuple (Name, Value, Type, TTL)
* Name and Value depend on Type
  + Type A : Normal host name to IP mapping
  + Type NS : Name is domain name and value is name of Authoritative server
    - It will then search for type A with name of authoritative server (To get which IP to go to).
  + CNAME : Value is Canonical hostname for alias hostname
  + MX : Value is canonical name for mail server that has an alias host name
    - After MX types, the query will search again for CNAME

DNS MESSAGE



* 12 byte header with a number of fields:
  + First 16 bits : Query Identifier
  + 1 bit to say if mesg is a query(0) or reply(1)
  + 1 bit to say if the reply is from Authoritative or not
  + 1 bit to tell it to perform recursion
* Question Section
  + Name that is being queried
  + Type of question being asked (A,NS….)
* Answer Section
  + Multiple answers
* Authority Section
  + Records of only the authoritative server
* Additional Section