software
studio

http:
hypertext transfer protocol

Daniel Jackson

For now, all rights reserved. Daniel Jackson, 2012.
what browser does

- basic page load is synchronous
- when page arrives, often loads others (images, css, js)
- complications: caching, asynchronous calls, events
http protocol

simplified protocol

```
session ::= request response
request ::= requestLine header+ [body]
requestLine ::= method path
response ::= status header+ [body]
```

what headers contain

› info about request or response, or about body

what body contains

› in request: form data, uploaded files
› in response: resource
GET, data passed as query string
Sample HTTP interactions

- POST, data passed as form data
response status codes

categories of codes
› 1xx informational
› 2xx success
› 3xx redirect
› 4xx client error
› 5xx server error

most common codes
› 200 OK (request succeeded, resource is in message body)
› 404 Not Found (resource doesn’t exist)
› 303 See Other (resource moved, see location header)
› 500 Server Error (web app implementer messed up)
http methods

› safe: no side effects
› idempotent: doing twice same as once
› PUT vs POST: whether object is at given URI, or child of
› PUT & DELETE used in APIs but not usually in browsers
http is stateless

HTTP
› server gets request
› performs action
› generates response

previous interactions
› have no effect on server
› same request, same action
why stateless?

no session state
› so memory doesn’t grow with number of clients
› if client dies, no problem for server

but...
› actually need session state (logged in, shopping cart)

solution
› server sends state to client as ‘cookie’
› client sends it back on each request

often
› server stores session state
› sends session id in cookie
cookies in http

cookie is
› name-value pair

server sends
› using set-cookie header

browser sends back
› all unexpired cookies
› with matching path

session cookies
› deleted when browser quits

persistent cookies
› have expiration date set

a funny cookie tale
nytimes.com used cookies to count #articles read, so viewers just deleted cookies...
example use of cookies: logging in

step 1: user opens home page
› request includes no cookies
› response body includes no member content

step 2: user submits login form
› request is POST with user and password as data
› response includes set-cookie headers <user: dnj, login: true>

step 3: user reopens home page
› request includes all cookies for domain
› response body includes member content

how to prevent cookies being faked?
› server encrypts cookie values with secret key
http is one-shot

implemented on TCP
› provides reliable transport
› large response message - many TCP messages
› but TCP suffers from slow start
› waste of effort to keep reopening same connection

persistent connections (HTTP 1.1)
› server keeps connection open
› client can pipeline requests before response received