

# CS290i - Lecture 3

## HTTP

Scalable Internet Services and Systems, Winter 2002

Thorsten von Eicken  
Department of Computer Science  
University of California at Santa Barbara

## HTTP 0.9

### ■ The Original HTTP as defined in 1991

- This document defines the Hypertext Transfer protocol (HTTP) as originally implemented by the World Wide Web initiative software in the prototype released. This is a subset of the full HTTP protocol, and is known as HTTP 0.9.
- No client profile information is transferred with the query. Future HTTP protocols will be back-compatible with this protocol.
- This restricted protocol is very simple and may always be used when you do not need the capabilities of the full protocol which is backwards compatible.
- The definition of this protocol is in the public domain (see policy).
- The protocol uses the normal internet-style telnet protocol style on a TCP-IP link.

## HTTP 0.9 (cont)

### ■ Connection

- The client makes a TCP-IP connection to the host using the domain name or IP number, and the port number given in the address.
- If the port number is not specified, 80 is always assumed for HTTP.
- The server accepts the connection.
- Note: HTTP currently runs over TCP, but could run over any connection-oriented service. The interpretation of the protocol below in the case of a sequenced packet service (such as DECnet(TM) or ISO TP4) is that the request should be one TPDU, but the response may be many.

## HTTP 0.9 (cont.)

### ■ Request

- A line of ASCII chars terminated by a CR LF pair.
- A well-behaved server will not require the CR.
- Request: "GET", a space, the document address.
- The document address will consist of a single word (e no spaces)
- The search functionality of the protocol lies in the ability of the addressing syntax to describe a search on a named index .
- A search should only be requested by a client when the index document itself has been described as an index using the ISINDEX tag.

## HTTP 0.9 (cont.)

### ■ Response

- The response is a message in HTML. This is a byte stream of ASCII characters.
- Lines shall be delimited by an optional CR followed by a mandatory LF.
- The format of the message is HTML. It also allows for plain ASCII text to be returned following the PLAINTEXT tag .
- The message is terminated by the closing of the connection by the server.
- Well-behaved clients will read the entire document as fast as possible. The server may impose a timeout of the order of 15 seconds on inactivity.
- Error responses are supplied in human readable text in HTML syntax. There is no way to distinguish an error response from a satisfactory response except for the content of the text.

5

## HTTP 0.9 (cont.)

### ■ Disconnection

- The TCP-IP connection is broken by the server when the whole document has been transferred.
- The client may abort the transfer by breaking the connection before this, in which case the server shall not record any error condition.
- Requests are idempotent. The server need not store any information about the request after disconnection.

### ■ Try:

```
[bugatti ~] telnet www.expertcity.com 80
Trying 63.251.224.189...
Connected to www.expertcity.com.
Escape character is '^]'.
GET /yadda<ret>
<ret>
```

6

## HTTP 1.0

### ■ URL = Uniform Resource Locator

- Resource ≠ file

### ■ Format of requests and responses

- Initial request/response line
- Zero or more header lines
- Blank line (CRLF)
- Optional message body

### ■ Initial request line:

- GET /path/to/file/index.html HTTP/1.0
- Other requests: POST, HEAD

7

## HTTP 1.0 (cont.)

### ■ Initial response line:

- HTTP/1.0 <code> <text>
- Code is error code:
  - 1xx: informational
  - 2xx: success, e.g. 200 OK
  - 3xx: redirect, e.g. 301 Moved Permanently, 302 Moved Temporarily
  - 4xx: error by client, e.g. 404 Not Found
  - 5xx: error by server, e.g. 500 Server error

### ■ Headers:

- <header-name>: <value>, <value>, ...
- Example:

```
Header1: some-value-1a, some-value-1b
HEADER1: some-value-1a,
some-value-1b
```

8

## HTTP 1.0 (cont.)

### ■ Message body

- The Content-Type: header gives the MIME-type of the data in the body, such as text/html or image/gif.
- The Content-Length: header gives the number of bytes in the body

### ■ HEAD request

- Returns same headers as GET, but no message body

### ■ POST request

- Send data to the server: message body
- Content-length & Content-type headers

9

## HTTP 1.0 Test

### ■ Try a GET request

```
[bugatti ~] telnet www.expertcity.com 80
Trying 63.251.224.189...
Connected to www.expertcity.com.
Escape character is '^]'.
GET /yasdda HTTP/1.0<ret>
<ret>
```

### ■ Try a HEAD request

```
[bugatti ~] telnet www.expertcity.com 80
Trying 63.251.224.189...
Connected to www.expertcity.com.
Escape character is '^]'.
HEAD /yasdda HTTP/1.0<ret>
<ret>
```

- (Actually read the headers of the response!)

10

## HTTP 1.1

### ■ Timeline

- Nov 95: first internet draft (before HTTP/1.0 draft)
- Jan 97: RFC2068 finished
- Nov 98: RFC2616 "draft standard" approved

### ■ New features

- Persistent connections
- Caching/proxy cleanup †
- IP address conservation (Host header)
- Partial transfers
- Content negotiation †
- Compression
- Digest authentication
- † not discussed in detail today

11

## Persistent Connections

- Connections are persistent by default:
  - ◆ Send request 1, receive response 1
  - ◆ Send request 2, receive response 2
  - ◆ Note: requires well-delimited requests & responses (content-length & chunked encoding)
- Termination:
  - ◆ Server: "Connection: close", or close connection (e.g. idle)
  - ◆ Client: close connection (or "Connection: close")
  - ◆ Note: client must handle aborted connections by retrying
- Pipelining:
  - ◆ send request 2 before response 1 recv'd
  - ◆ Strictly in-order responses
  - ◆ Careful with non-idempotent requests

12

## Host+ Range header

### Host header

- Problem: 1 server but 1000 domain names (www.thorsten.com)
- Solution: host header is required
  - ◆ GET /index.html HTTP/1.1  
Host: www.thorsten.com
  - ◆ As opposed to assigning 1000 IP addresses to the server

### Range header

- Problem: aborted connections, fetching prefixes
- Solution: specify requested byte range
  - ◆ GET /bigimage.jpg HTTP/1.1  
Range: bytes=12345-

### HTTP 1.1 Test:

- try to "GET / HTTP/1.1", then try keep-alive

13

## Jeff Mogul's gripes



### Who's this?

- Compaq (was Digital) Western Research Lab
- Active in HTTP design, web server performance,

### Gripes

- Data model, MIME miasma
- Extensibility
- Caching
- Header categories
- Status and error codes
- Transport issues
- Content negotiation
- Cookies & social issues



14

## Data Model & MIME

- HTTP-OO "vision":  
bogus from the start...
  - ◆ HTTP transfers current response, NOT resource itself
  - ◆ Cached values don't work like dynamic resources
  - ◆ No cache coherency for updatable resources
- Result
  - ◆ HTTP/1.1 no longer described as O-O
  - ◆ So, now what?

**Data model and the MIME miasma**

**Original HTTP vision:**

- "object-oriented protocol"
- MIME-conforming and MIME-compatible
- Objects with many variants

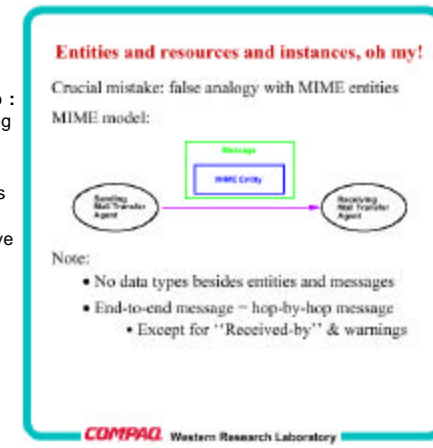
**Consequences:**

- Terminology:
  - "Objects" and "methods"
  - MIME "entity" for in-transit payload
- Possibility of "dynamic" resources
- MIME header rules (more or less)
- MIME content-type system
- URL does not always identify specific variant

15

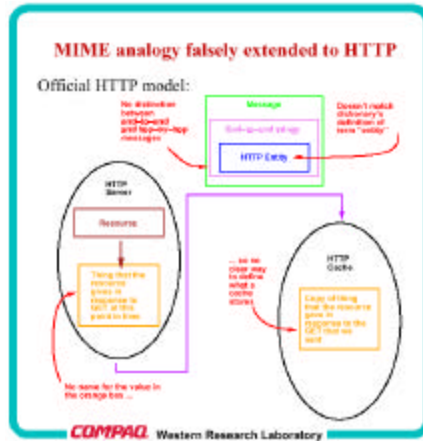
## Data Model & MIME (cont.)

- #18: "Entity":
  - ◆ 1 a : independent, separate, or self-contained existence **b** : the existence of a thing as contrasted with its attributes;
  - ◆ 2 : something that has separate and distinct existence and objective or conceptual reality



16

## Data Model & MIME (cont.)



17

## Data Model & MIME (cont.)

- Example: What is the server supposed to respond to a request with:

- Content-Type: text/plain
- Content-Encoding: compress
- Transfer-Encoding: gzip, chunked
- Range: bytes=1024-2047

### Data model - importance

Why is this important?

- Poor terminology leads to fuzzy thinking
- Poor terminology leads to underspecification
  - E.g., relationship between compression and byte-ranges
- Much confusion over whether headers apply to:
  - Server (or proxy)
  - Resource
  - Variant
  - Instance
  - Entity
  - End-to-end message
  - Hop-by-hop message
 with some headers filling several roles

Note: MIME's content-type system is fine with me.

COMPAQ Western Research Laboratory

18

## HTTP types/codings

- §3.5: "Content coding values indicate an encoding transformation that has been applied to an entity. Content codings are primarily used to allow a document to be compressed or otherwise usefully transformed without losing the identity of its underlying media type and without loss of information. Frequently, the entity is stored in coded form, transmitted directly, and only decoded by the recipient."
- §3.6: "Transfer-coding values are used to indicate an encoding transformation that has been applied to an entity-body in order to ensure "safe transport" through the network. This differs from a content coding in that the transfer-coding is a property of the message, not of the original entity."
- §7.2.1: "When an entity-body is included with a message, the data type of that body is determined via the header fields Content-Type and Content-Encoding. These define a two-layer, ordered encoding model:
 

```
entity-body := Content-Encoding( Content-Type( data ) )
```

 Content-Type specifies the media type of the underlying data. Content-Encoding may be used to indicate any additional content codings applied to the data, usually for the purpose of data compression, that are a property of the requested resource."
- §14.35.1: "Byte range specifications in HTTP apply to the sequence of bytes in the entity-body (not necessarily the same as the message-body)."

19

## HTTP types/codings (cont.)

### Questions:

- In the following scenarios, what should your server send and what headers should it use?
- 1: request to transfer bytes 1024-2047 of a file called foo.html.gz
- 2: request to transfer bytes 1024-2047 of a file called foo.html and the client indicates it can handle a gzip transfer-coding
- 3: request to transfer bytes 1024-2047 of foo.html.gz when only foo.html exists, but your server is "smart" and can gzip on the fly
- 4: request to transfer bytes 1024-2047 of foo.gif when only foo.jpg exists, but your server is "smart" and can transform on the fly

20

## Extensibility

### ■ Facts of life

- HTTP evolves
- Compatibility is mandatory
- “Flag days” are impossible

### ■ So: HTTP must be extensible

- Interoperate with past implementations
  - ◆ Without degrading their behavior
- Interoperate with future implementations
  - ◆ No matter what the future brings
- Ambiguity kills

### ■ Three key issues

1. Protocol version numbering
2. Probing for extensions
3. The POST hole

21

## Extensibility: version nums

- Multi-version chains are possible:  
HTTP/1.0 200 OK  
Via: 1.1 fred, 1.0 lucy
- For example, byte ranges are optional

**HTTP Version numbers**

HTTP version numbers verge on meaningless

- “HTTP/1.0” never really specified
- “HTTP/1.1” systems deployed before spec is finished
- Some proxies use the wrong version number
- Version number is officially hop-by-hop
  - ◆ though Via gives end-to-end version
- Optional features aren’t indicated by version

Basic problems:

- Distinction between hop-by-hop, end-to-end
- Lack of precision
  - ◆ optional features (MUST vs. SHOULD)
  - ◆ minor updates
  - ◆ “Proposed” vs. “Draft” vs. “standard”
- Lack of formal binding to a specification

COMPAQ Western Research Laboratory

22

## Extensibility: probing

- Question:
  - ◆ Does peer implement an extension?
- Goals:
  - ◆ Discover this reliably
  - ◆ Fall back to standard protocol
  - ◆ Don’t add too many round trips
- One good thing: “ignore unknown headers” rule
  - ◆ See if peer responds to new header
  - ◆ Especially useful for optimizations
- Complex extensions not adequately supported
  - ◆ What precise extension (and options) is desired?
  - ◆ How does this affect caching?
  - ◆ How are extensions named?
- Attempts:
  - ◆ PEP (“Protocol Extension Protocol”): failed
  - ◆ “HTTP Extension Framework”: jury is still out

23

## Extensibility: the POST hole

- HTTP’s POST method
  - ◆ Sends bits to server
  - ◆ ... but does not simply store into resource
  - ◆ Basically, an arbitrary RPC
  - ◆ Typical use: complex HTML forms
- A well-specified standard would ...
  - ◆ allow ends to evolve without agreement
  - ◆ allow proxies to intermediate
- POST is just a big loophole
  - ◆ No standard
  - ◆ Too tempting to use instead of a true extension
  - ◆ Problematic for security firewalls
- Byte way...
  - ◆ when to GET vs. POST?

24

# Caching

## Cache locations:

- Server
- Public proxies
- Private proxies
- User-agent (browser)

### Caching

#### Caching and proxies in HTTP/1.0:

- An afterthought, at best
- No cache coherency (especially for updates)
- No extensibility to new methods
- No detection of transparency failures

#### Transparency:

A cache behaves **transparently** when the response that you get from it is essentially the same as the response you would have received from the origin server.

#### Without transparency:

- Caches aren't trusted (and are bypassed)
- Users get wrong answers

# Caching, how it works

## How to detect updates?

- Conservative expiration deadlines
- Do many requests to server anyway

## Consistency & security

- Can  
//www.expertcity.com/index.html invalidate  
//63.251.224.189/index.html?

### How HTTP caching works

#### Basic model of HTTP caching:

1. Client A requests resource R via Proxy P
2. Proxy P forwards A's request to server for R
3. Proxy P receives server's response for R
  - Forwards response to A
  - Stores response for later use
4. Client B requests resource R via Proxy P
5. Proxy P retrieves & returns stored response

#### Conditional requests:

1. Server's response includes **validator**
  - e.g., last-mod date or (in 1.1) entity-tag
2. Client asks server if cache entry is fresh:  
`GET /foo.gif HTTP/1.0`  
`If-Modified-Since: Thu, 03 Jun 1999 20:16:34 GMT`
3. Server responds with either:
  - 200 OK + full response body
  - 304 Not Modified + no body

# Caching, complexity

- Age calculation, expiry date, last-modified date, "max-age=3600", if-modified-since

## Variants:

- request "Accept-Language: fr, en;q=0.5"
- got en-br version in cache
- can it be returned?

### Caching - complexity



#### HTTP/1.1 caching specification:

- Lots of caching-related rules
- Some are subtle
- Some are contradictory

#### This is due to

- Evolutionary design
- Competing notions of goodness

#### Cache implementations are complex:

- Because of complex specification
- Because of complex lookup mechanisms
- Because of inevitable engineering issues

Not clear how to resolve this

# Caching, is it worth it?

### Caching - performance

#### Three reasons for caching

1. Faster response time
2. Lower bandwidth requirements (\$\$\$)
3. Availability during disconnection

#### But

- Observed cache hit ratios are below 50%
- Byte-weighted hit ratios are even lower
- Lack of coherent "collection" concept

So **simple caching has pretty much hit its limits**

Can't we do better?

## Content negotiation

- Request headers
  - ◆ Accept[+type]
  - ◆ Accept-Charset
  - ◆ Accept-Encoding
  - ◆ Accept-Language
  - ◆ Upgrade
  - ◆ Accept-Encoding applies to Content-Encoding, not Transfer-Encoding! (TE is for latter, so what's the point?)
- Response headers
  - ◆ Accept-Ranges
  - ◆ Connection

**Content negotiation**

**Important original broad goal:**

- Make the Web international (multilingual)

**Semi-failure, because of fuzziness about:**

- Specific goals; especially, who's in charge?
  - User's preference
  - Site designer's preference
- Naming issues
- Confusion between negotiation axes:
  - content
  - presentation
  - implementation parameters
- Caching
- Importance of avoiding round-trips

COMPAQ Western Research Laboratory

29

## Status codes

- HTTP responses carry one 3-digit status code; e.g.
  - ◆ HTTP/1.0 200 OK
  - ◆ HTTP/1.0 304 Not Modified
  - ◆ HTTP/1.0 404 Not Found
  - ◆ HTTP/1.0 501 Not Implemented
- Problems:
  - ◆ Some ambiguity in specification
  - ◆ "Which code should I use here?"
  - ◆ Only one code
  - ◆ What about non-fatal errors at proxies?
  - ◆ Complex interaction with caching
  - ◆ Not really extensible

30

## Transport issues

- Goal: efficient and reliable message transport
- Issues:
  - ◆ Inefficiency of ASCII header encoding
  - ◆ Bias against compression
  - ◆ Lack of clean connection-abort mechanism
  - ◆ No buffer size limits/negotiation
  - ◆ No multiplexing
  - ◆ No multi-resource operations
  - ◆ No atomic grouping of operations
- HTTP/1.1 has some improvements
  - ◆ Persistent connections, pipelining
  - ◆ Compression negotiation
  - ◆ Chunked encoding + careful rules
- Sample issues:
  - ◆ max size of a URI? Of headers? Failure indication?

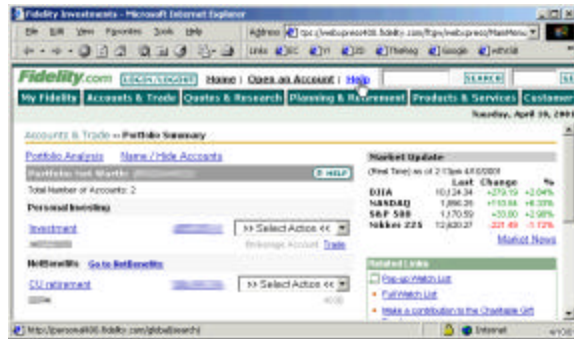
31

## Cookies & social issues

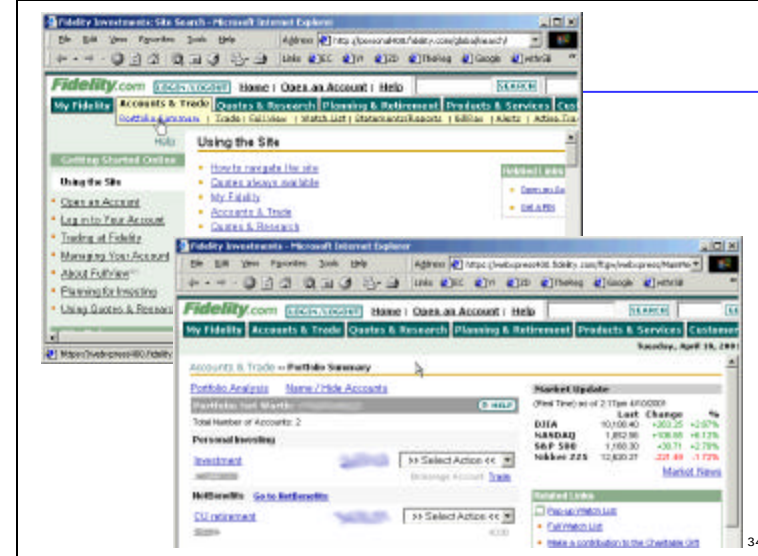
- **Conflict between technology, policy, & profit**
  - IETF requires "Security Considerations"
  - but there's no consensus on how far to go
  - Ad-supported sites have a lot to lose
  - Software vendors tend to play along
- **Cookies not part of HTTP/1.1! See RFC2965...**
  - Headers: Set-Cookie2, Cookie
  - Fields:
    - ◆ Domain=.expertcity.com
    - ◆ Path=/myaccount/
    - ◆ Port="80,8080"
    - ◆ Discard (when u-a terminates)
    - ◆ Max-age=3600
    - ◆ Secure

32

# Cookies & security

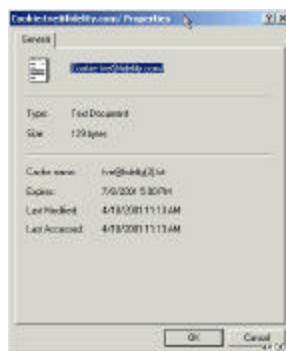


33



34

# Cookie: what's in it?



35

# An example

- `snoop -c 30 -s 1500 -d /hml -r -V -x 0 port 80 and host 206.72.82.14`
- `GET /images/ad/gt2lp.html HTTP/1.1`  
**Accept:** \*/\*  
**Accept-Language:** en-us,fr-ch;q=0.8,ca;q=0.6,de;q=0.4,es;q=0.2  
**Accept-Encoding:** gzip, deflate  
**User-Agent:** Mozilla/4.0 (compatible; MSIE 5.5; Windows NT5.0; T312461)  
**Host:** img.gotomypc.com  
**Connection:** Keep-Alive  
**Cache-Control:** no-cache
- `HTTP/1.1 200 OK`  
**Date:** Wed, 16 Jan 2002 17:58:41 GMT  
**Server:** Apache/1.3.11  
**Last-Modified:** Fri, 11 Jan 2002 16:40:06 GMT  
**ETag:** "71bbe-1cc-3c3f1566"  
**Accept-Ranges:** bytes  
**Content-Length:** 460  
**Keep-Alive:** timeout=3, max=100  
**Connection:** Keep-Alive  
**Content-Type:** text/html

36

## Example (cont.)

```
■ GET /images/ad/g2_gator_pu.gif HTTP/1.1
Accept: */*
Referer: http://img.gotomypc.com/images/ad/gt2lp.html
Accept-Language: en-us,fr-ch;
...
Cache-Control: no-cache

■ HTTP/1.1 200 OK
Date: Wed, 16 Jan 2002 17:58:42 GMT
Server: Apache/1.3.11
Last-Modified: Wed, 09 Jan 2002 22:43:53 GMT
ETag: "71bbc-779e-3c3cc7a9"
Accept-Ranges: bytes
Content-Length: 30622
Keep-Alive: timeout=3, max=99
Connection: Keep-Alive
Content-Type: image/gif
```

37

## Summary

### ■ Why is it so hard?

- Each detail is simple
- The collection is unmanageable
- Need more layers? More orthogonality?

### ■ How can you implement it?

- What do you expect clients to implement?
- What do you expect servers to implement?
- What do you expect proxies to implement?
- What do you expect all these to get wrong?
- How do you test your own implementation?

### ■ How come it works at all?

- Nobody uses the full spec?
- The real spec is IE & NS?

38