HTTP

Internet Engineering

Spring 2014

Bahador Bakhshi

CE & IT Department, Amirkabir University of Technology

Questions

- Q1) How do web server and client browser talk to each other?
- Q1.1) What is the common protocol?
- Q1.2) How are resources identified?
- Q1.3) What are requests & responses?
- Q1.4) Can/Should server know its clients?
- Q1.5) Who can influence the communication between server & client?
- Q1.6) Is everything public?

Homework

- HW-1: HTTP Analyzer program
  - Connect to a URL & extract HTTP information
- Requirements
  - Socket Programming
  - HTTP
- Deadline
  - At least one week after finishing this lecture
  - Exact time will be announced

Outline

- Introduction
  - Messages
  - Methods
  - Headers
- Cookie
- Proxy
- Cache
- Authentication

Outline

- Introduction
  - Messages
  - Methods
  - Headers
- Cookie
- Proxy
- Cache
- Authentication

Introduction

- The transfer protocol for web applications is HTTP
  - HTTP: Hyper Text Transfer Protocol
  - HTTP 1.0 (RFC 1945), HTTP 1.1 (RFC 2068)
  - Can be used to transfer everything
    - Text documents: HTML, XML, …
    - Multimedia: JPG, GIF, Video, …
- HTTP uses the client/server paradigm
  - HTTP server provides resource
  - HTTP client (usually web browser) gets resource
- But not pure client/server communication
  - Proxies, caches, …

Introduction (cont'd)

- HTTP is an application layer protocol
- HTTP assumes reliable communication
  - TCP, default (server) port: 80
- HTTP is stateless
  - Server does not keep history/state of clients
    - If client asks an object 10 times, server will give it back each time
  - High performance & Low complexity
  - Problematic in some applications (sessions)
    - Cookies or other solutions

Resources

- HTTP is the protocol to transfer data between server and client (usually from server to client)
- Which data?
  - It can be anything
  - In web, usually, it is a resource/object on server
- Each resource must be identified uniquely
  - URI (Uniform Resource Identifier)
- Common practical implementation of URI is URL
  - Uniform Resource Locator

URL

- URL requirements
  - Basic requirements:
    - Destination machine identification
    - Transport layer port identification
    - Application layer protocol identification
    - Resource address in the destination machine
  - Additional requirements:
    - Security/Authentication
    - Sending data from client to server
    - Partial access to resource

URL (cont'd)

- <protocol(scheme)>://<user>:<pass>@<host>:<port>/<path>?<query>#<frag>

- Some examples:

  http://www.aut.ac.ir

  ftp://me:123@kernel.org/pub

  http://www.bing.com/search?q=web&go=&qs=n&form=QBLH&pq=web&sc=0-0&sp=-1

  file://c:/windows/ file:///home/bahador/work

URL (cont'd)

- *Scheme*: the application layer protocol
- HTTP: The web protocol
- HTTPS: Secure HTTP
- FTP: File transfer protocol
- File: Access to a local file
- mailto: Send email to given address
- javascript: Run javascript code
- …

URL (cont'd)

- *Path*: the path of the object on host filesystem
  - It is with respect to web server (document) root directory
- E.g. web server root directory: /var/www/
  - http://www.example.com/1.html
    - /var/www/1.html
  - http://www.example.com/1/2/3.jpg
    - /var/www/1/2/3.jpg
- Similar to filesystem paths
  - Absolute: Path starts from web server root directory
  - Relative: Path starts from current directory

URL (cont'd)

- *Query*: a mechanism to pass information from client to active pages or forms
  - Fill information in a university registration form
  - Ask Google to search a phrase
- Starts with "?"
- "&" is the border between multiple parameters

http://www.example.com/submit.php?name=ali&family=karimi

URL (cont'd)

- *Frag*: A name for a part of resource
  - A section in a document

http://www.example.com/paper.html#results

- Handled by browser
  - Browser gets whole resource (doc) from sever
  - In display time, it jumps to the specified part

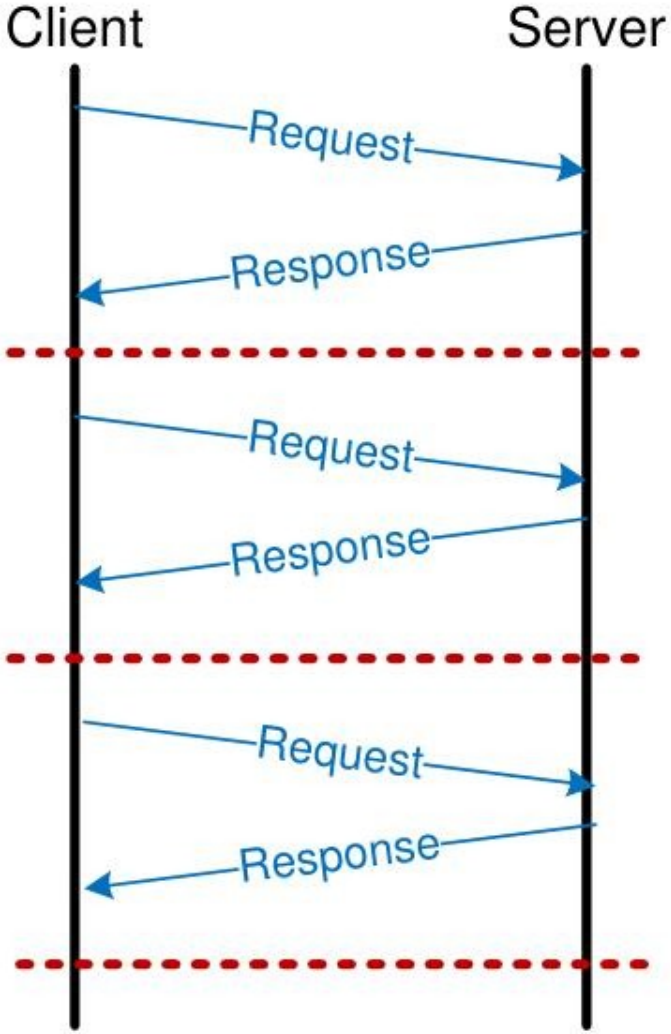
URL (cont'd)

- URL is encoded by client before transmission
- How: Each byte is divided into two 4-bit group, hexadecimal of the 4-bits are prefixed by %
  - Example: ~ → 126 (ASCII) → 01111110 → %7E
- What & Why?
  - Non-ASCII (e.g., Persian characters)
  - ASCII control because is not printable
  - Reserved character when are not used for special role
    - E.g. @, :, $, …
  - Unsafe character, e.g. space, %, …

HTTP Transaction

- HTTP data transfer is a collection of transactions
- Transactions are performed by HTTP messages
  - Client → Server: HTTP Request message
  - Server → Client: HTTP Response message
- Requests are identified by methods
  - Method: The action that client asks from server
- Response are identified by status codes
  - Status: The result of the requested action

HTTP Transaction (cont'd)



HTTP Transaction in Web

- (Typically) each web page contains multiple resources
  - The main skeleton HTML page
  - Some linked materials: figures, videos, JS, CSS, …
- Displaying a web page by browser
  - Get the HTML page (first transaction)
  - Try to display the page (rendering)
    - Other resources are linked to the page
  - Get the resources (subsequent transactions)

HTTP Transaction in Web

- HTTP Transactions & TCP connections
- 1) Non-persistent
  - A new TCP connection per object
    - Networking overhead + Connection establish delay + Resource intensive (specially in server side)
    - Parallel connections speed up browsing
- 2) Persistent
  - Get multiple objects using single TCP connection
    - No processing & networking overhead
    - Very poor performance if implemented in serial manner
      - Pipeline requests speed up browsing

HTTP Transaction in Web: Example

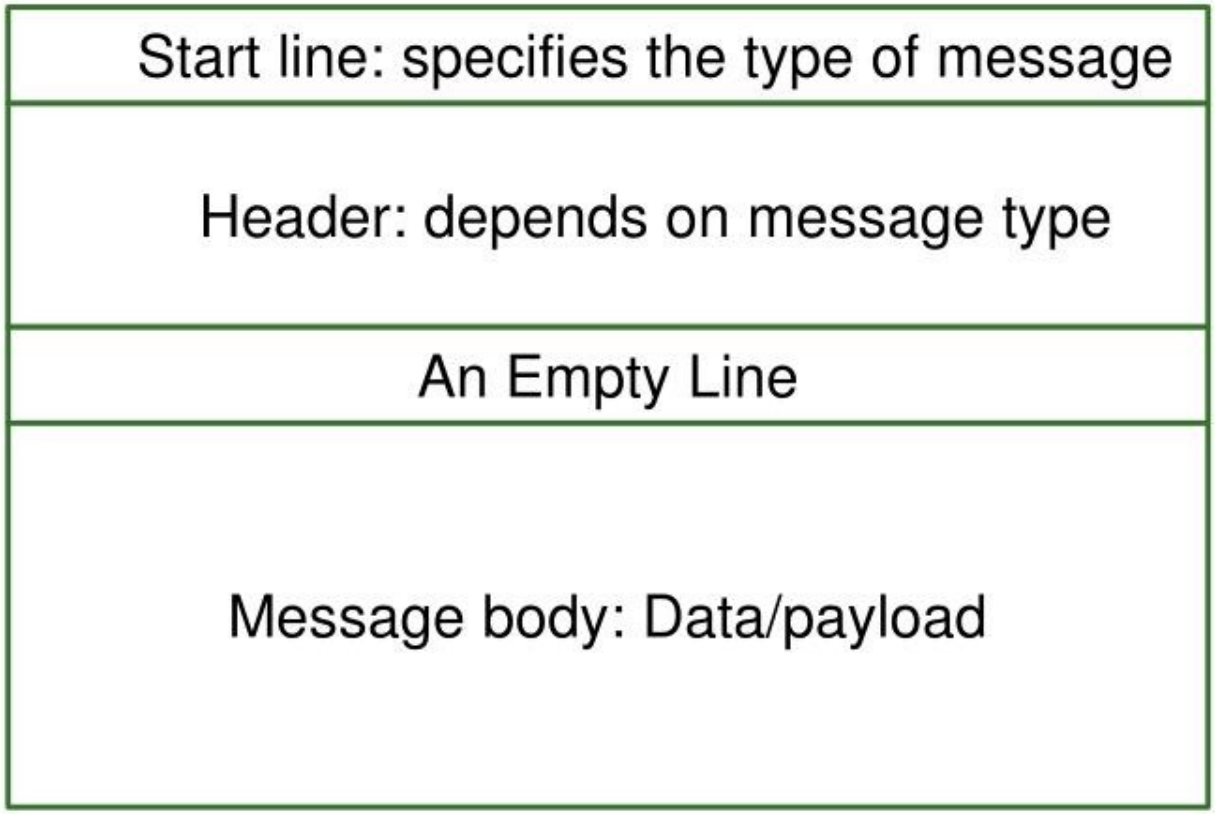
- Get a HTML page from a server
- Capture the packets
- Investigate the transactions

Outline

- Introduction
  - Messages
  - Methods
  - Headers
- Cookie
- Proxy
- Cache
- Authentication

HTTP Messages

- Messages (request/response)



HTTP Messages (cont'd)

- Request message format

Method<sp>Path<sp>version<CRLF>

<Header field>:<value><CRLF>

...

<Header field>:<value><CRLF>

<CRLF>

<Entity body>

HTTP Messages (cont'd)

- E.g. HTTP request message

```
GET /index.html HTTP/1.1
Host: www.aut.ac.ir
User-Agent: Mozilla/6.0
Accept-Language: en-us
Connection: keep-alive
```


HTTP Messages (cont'd)

- Response message format

Version<sp>code<sp>Reason<CRLF>

<Header field>:<value><CRLF>

...

<Header field>:<value><CRLF>

<CRLF>

<Entity body>

HTTP Messages (cont'd)

- E.g. HTTP response message

HTTP/1.1 200 OK

Date: Sun, 02 Oct 2011 20:30:40

Server: Apache/2.2.2

Last-Modified: Mon, 03 May 2009 10:20:22

Connection: keep-alive

Content-Length: 3000

(data data data ….)

HTTP Methods

- Methods are actions that client asks from server to do on the specified resource (given by the path parameter)
- Which actions?
  - Basic data communication operations
    - Safe operations
      - Get a resource from server
      - Send data to server
    - Unsafe operations
      - Delete a resource on server
      - Create/Replace a resource on server
  - Debugging and troubleshooting
    - Get information about a resource
    - Check what server got from client
    - Get List of operations which can be applied on a resource

HTTP Methods (cont'd)

- GET (must be implemented by server): Retrieve resource from server
- HEAD (must be implemented by server): Similar to GET but the resource itself is not retrieved, just the HTTP response header
  - Useful for debugging or some other applications

HTTP Methods (cont'd)

- POST: Submit data to be processed by the specified resource
  - Data itself is enveloped in message body
- DELETE: Remove the resource!!!
- PUT: Add message body as the specified resource to server!!!
- TRACE: Server echoes back the received message
  - For troubleshooting & debugging

HTTP Methods (cont'd)

- OPTIONS: Request the list of supported methods by server on the resource
- CONNECT: Create HTTP tunnel
  - Client asks server (which is proxy/gateway) to create TCP connection to the specified destination
  - After TCP connection establishment, all data sent on TCP connection between client & server are copied to the established new TCP connection

HTTP Methods (cont'd): Examples

- Connect to a web server
  - telnet can create TCP socket
- Play with the server by sending HTTP methods and checking the responses

HTTP Responses

- The message for the result/response of the requested action
- Which responses?
  - Basic responses
    - Success
    - Failure
      - Bad client request
      - Server problem
  - Redirection to other resources

HTTP Responses (cont'd)

- 2xx: Successful operation
  - 200: OK
  - 201: Created
- 3xx: Resource has been moved, Redirection
  - Location header → the new location of resource
  - 301: Moved Permanently
  - 304: Not modified
  - 307: Moved Temporarily

HTTP Responses (cont'd)

- 4xx: Client error
  - 400: Bad request
  - 401: Unauthorized (Authorization required)
  - 403: Forbidden
  - 404: Not found
  - 405: Not allowed method
- 5xx: Server error
  - 500: Internal server error
  - 501: Not implemented
  - 503: Service unavailable

HTTP Headers

- Headers are additional information sent by client to server and vice versa
  - Almost all are optional
- Which headers?
  - Information about client
  - Information about server
  - Information about the requested resource
  - Information about the response
  - Security/Authentication
  - …

HTTP Headers

- General headers
  - Appear both on request & response messages
- Request headers
  - Information about request
- Response headers
  - Information about response
- Entity headers
  - Information about body (size, …)
- Extension headers
  - New headers (not standard)

General Headers

- Date: date & time that *message* is created
- Connection: close or keep-alive
  - Close: Non-persistent connection
  - Keep-alive: Persistent connection
- Via: Information about the intermediate nodes between two sides
  - Proxy servers

Request Headers

- Host: The name of the server (required, why?)
- Referer: URL that contains requested URL
- Information about client
  - User-Agent: The client program
  - UA-OS: The OS of client program
  - UA-Disp: Information about display of client
  - Accept: The acceptable media types
  - Accept-Encoding: Acceptable encoding
  - Accept-Language: What language are acceptable

Request Headers (cont'd)

- If-Modified-Since: Request is processed if the objected is modified since the specified time
- Authorization: Response to the authenticate requests
- Range: Specific range (in byte) of resource

Response Headers

- Server: Information about server
- WWW-Authenticate: Used to specify authentication parameters by server
- Proxy-Authenticate: Used to specify authentication parameters by proxy
- Set-Cookie: To send a cookie to client

Entity Headers

- Content-Length: The length of body (in byte)
- Content-Type: The type of entity
  - MIME types: text/html, image/gif
- Allow: The allowed request method can be performed on the entity
  - This is in response of OPTIONS method
- Location: The new location of entity to redirect client

Entity Headers (cont'd)

- Content-Range: Range of this entity in the entire resource
- Expire: The date and time at which the entity will expire
- Last-Modified: The date and time of last modification of entity
- ETag: A tag for entity (usually time based)
- Cache-Control: To control entity caching

Outline

- Introduction
  - Messages
  - Methods
  - Headers
- Cookie
- Proxy
- Cache
- Authentication

Stateless Problem

- HTTP is a stateless protocol
  - Server does not remember its clients
- How to personalize pages (personal portal)?
  - Use http header: Client-ip, From, …
    - Is not usually sent by browsers
  - Find client IP address from TCP connection
    - The problem is NAT

Solution of Stateless Problem: Cookie

- Cookies: Are unique identifiers assigned by server to each user (*browser*) or session
- How it works
  - Server asks client to remember the ID
    - Set-Cookie header in response message
  - Client gives back the ID to server in *every* request
    - Cookie header in request messages
  - Server customizes responses according to cookie
- Types
  - Session cookies: To identify a session
  - Persistent cookies: To identify a client (browser)

Cookies (cont'd)

1)

```
GET /index.html HTTP/1.1
Host: www.example.org
```

browser --------→ server

2)

```
HTTP/1.1 200 OK
Content-type: text/html
Set-Cookie: name=value
Set-Cookie: name2=value2; Expires=Wed, 09 Jun 2021 10:18:14 GMT

(content of page)
```

browser ←-------- server

3)

```
GET /spec.html HTTP/1.1
Host: www.example.org
Cookie: name=value; name2=value2
Accept: */*
```

browser --------→ server

Cookies (cont'd)

- Limitation (cannot be used to store large data)
  - Typically 300 total cookies, 20 cookies per domain, 4KB data per cookie
- Cookies are text files
  - No virus spread
- There is not any request from server to read cookies
  - By default cookies are sent by browser
  - Browser checks URL and finds appropriate cookies

Cookies (cont'd)

- Client can control cookies
  - Disable cookies: no cookie is saved & used
  - View & Delete cookies
- Server can control cookies by its attributes
  - Expiration time
  - Domain
  - Path
  - Security
  - …

Cookies Attributes

- Expire & Max-Age: The life time of the cookie
  - Expire: An absolute time to delete cookie
  - Max-Age: The maximum life time (sec) of cookie
  - Send a past time (or negative) to delete a cookie
- Secure: Cookie is sent only if channel is secure
  - Specially useful for cookies for login sessions
- HttpOnly: Cookie is sent only if HTTP is used
  - JavaScript cannot access to the cookies

Cookies Attributes: Domain & Path

- Domain & Path determine the scope of the cookie
  - For which path and domain, the cookie is saved & returned back by browser
- If these attributes are absent → browser assumes current *host* & current *path*
  - Browser returns back the cookie for the host & path and also for all sub-paths
- If are present → browser checks validity
  - If they are valid → Browser returns back the cookie for that *domain* & that *path* and also for all *subdomains* and *sub-paths*

Cookies Attributes: Domain & Path

- Validity check by major browsers
  - Domain names must start with dot
    - Some browsers accept names without dot as domain
  - Don't accept cookies for sub-domains
  - Accept cookies for higher domains
    - Except the top level domains, e.g., .com, .ac.ir
  - Accept cookies for other (sub or higher) paths

Cookies Attributes: Domain & Path Examples

- A php script sets some cookies
- The script can be run using different domains
- Check out which cookies are accepted
- Check out which cookies are sent back

How are Cookies Stored in Practice

- How cookies are saved in browser?
- Example: Firefox

```
cd ~/.mozilla/firefox/xyzxxxx/
sqlite3 cookies.sqlite
.tables
.schema <the table>
select * from <the table>;
```

Outline

- Introduction
  - Messages
  - Methods
  - Headers
- Cookie
- Proxy
- Cache
- Authentication

Proxy

- Proxies sit between client and server
- Act as server for client
- Act as client for server

HTTP Proxy Applications

- Authentication
  - Client side: Authenticate clients before they access web
  - Server side: Authenticate clients before they access the server
- Accounting: Log client activities
- Security: Analyze request before send it to server
  - Integrated in modern firewalls
- Filtering: Limit access to specified contents
- Anonymizer: Anonymous web browsing
- Caching (more details in the following slides)

Proxy in Action

- How to redirect traffic to proxy:
- 1) Client configuration
  - Manual configuration
  - Automatic (WPAD protocol) & Scripting
- 2) L4/7 switches
  - Redirect traffic according to destination port
- 3) DNS mechanisms
  - Return proxy server's IP address instead of server's address

Outline

- Introduction
  - Messages
  - Methods
  - Headers
- Cookie
- Proxy
- Cache
- Authentication

Caching

- Caching: save a copy of a resource and use it instead of requesting server
- Browser has its own local caches
- Cache server is special proxy for caching
- Benefits
  - Reduce redundant data transfer
  - Reduce network bottleneck
  - Reduce load on server
  - Reduce delay

Caching (cont'd)

- Cache cannot keep copy of whole objects (why?!)
- Two possible cases for requested objects
  - Hit: Cache has a fresh copy of the object
  - Miss: Otherwise
- Cache performance index
  - Hit ratio (0~1)
    - Object hit ratio vs. Byte hit ratio
      - Higher Object hit ratio → More responsive browsing
      - Higher Byte hit ratio → More bandwidth saving
    - Depends on Object replacement algorithm, Object selection algorithm, Cache size, Similar activities, Objects size, Objects type, Caching control headers, …

Caching Algorithm

- If the object is not cached, it is got from server, saved in cache (if it is cacheable), and sent to client
- Else, if object is in cache
  - Cache server must return only fresh objects
    - Freshness check
- Objects life-time specified by server
  - Expire header: Absolute expiration time
  - Cache-Control: max-age: Relative expiration time
- If requested object is not expired
  - Cache server gives it to client

Caching Algorithm (cont'd)

- If requested object is expired
  - Its freshness must be checked
- Freshness is checked by conditional request
  - If-Modified-Since: current last-modified time
  - If-None-Match: current associated Etag
- Server responses
  - 304 Not modified response + new expire time
    - Cached copy is valid until the specified time
  - 200 OK
    - Server provides a new version of the object
    - Cache server updates cached copy

Caching Control

- Caching can be controlled by both client & server
- Client can force to not to use cached objects using the Cache-Control header
  - min-fresh: cached object that will be fresh until given time
  - max-age: cached object not cached time more than given time
  - max-stale: maximum age of not fresh cached object
  - no-cache: no cached object
- Server has control using Cache-Control header
  - no-cache: do not cache this object
  - must-revalidate: Check freshness every time
  - max-age: the relative expiration time
  - Beside Cache-Control, the Expire header is used

Outline

- Introduction
  - Messages
  - Methods
  - Headers
- Cookie
- Proxy
- Cache
- Authentication

HTTP Authentication

- All resources are **not** public in web; e.g.,
  - Financial documents
  - Customer information
  - Portals
- HTTP has two (similar) authentications
  - Basic: Base64 encoded “user:pass”
  - Digest: Plain username + Digest of pass
- Steps are the same
  - Server challenges, Client Responses, Server authenticates

HTTP Authentication

Basic authentication

1) Client → Server

```
GET /private/index.html HTTP/1.1
Host: localhost
```

2) Server → Client

```
HTTP/1.1 401 Authorization Required
Server: HTTPd/1.0
Date: Sat, 27 Nov 2004 10:18:15 GMT
WWW-Authenticate: Basic realm="Secure Area"
Content-Type: text/html
Content-Length: 311
```

3) Client → Server

```
GET /private/index.html HTTP/1.1
Host: localhost
Authorization: Basic QWxhZGRpbjpvcGVuIHNlc2FtZQ==
```

2) Server → Client

```
HTTP/1.1 200 OK
Server: HTTPd/1.0
Date: Sat, 27 Nov 2004 10:19:07 GMT
Content-Type: text/html
Content-Length: 10476
```

Digest Authentication

- Basic authentication is insecure
  - Password is sent in base64 encoding
  - Attacker can easily find it
- Digest authentication: Don't send password
  - Send its digest (hash)
- Digest/hash function
  - One way function, irreversible
- Attacker cannot find password ☺
- But! Reply attack ☹
  - Attacker resends the same digest → He is authenticated
  - Use Nonce

Digest Authentication

- 1) Client requests a private resource
- 2) Server creates a nonce
  - WWW-Authenticate: Digest nonce=39X9s#! …
- 3) Client computes digest of password and nonce
  - Authorization: username, hash(pass, nonce)
- 4) Server looks up the password of username and computes hash(pass, nonce)
  - If this value and Authorization are the same → Ok

Real Example

- Basic & Digest Authentication using the .htaccess mechanism in Apache server

Security

- Digest authentication protect password
- Data is completely insecure
- No mechanism in HTTP to protect data
- HTTP over SSL/TLS is a popular solution
  - An encrypted tunnel between client & server
  - Send HTTP traffic through the tunnel

Answers

- Q1.1) What is the common protocol?
  - HTTP, Message = Header + Body
- Q1.2) How are resources identified?
  - URL
- Q1.3) What are requests & responses?
  - Method (GET, HEAD, …)
  - Status (2xx, 3xx, 4xx, 5xx)
- Q1.4) Can/Should server know its clients?
  - Yes, using cookies
- Q1.5) Who can influence the communication between server & client?
  - Proxy servers, e.g., cache servers
- Q1.6) Is everything public in server?
  - No, HTTP basic/digest authentication

Homework

- Deadline: 23:59 1392/12/2
- Submit to Moodle
  - Email otherwise

Reference

- HTTP: The Definitive Guide

\\fileserver\common\Bakhshi\Internet Engineering\