Network programming (IT423+IT432)

Spring 2017

Dr. Islam Taj-Eddin

IT Dept., FCI, Assiut Univ.

HTTP

The HTTP protocol

- The HTTP protocol is the standard protocol for communication between web browser and web server.
- HTTP specifies:
  - how a client and server establish a connection,
  - how the client requests data from the server,
  - how the server responds to that request,
  - and finally, how the connection is closed.

What is a protocol?

- In diplomatic circles, a protocol is the set of rules governing a conversation between people
- We have seen that the client and server carry on a machine-to-machine conversation
- A network protocol is the set of rules governing a conversation between a client and a server
- There are many protocols, HTTP is just one

HTTP connections use the TCP/IP protocol for data transfer. For each request from client to server, there is a sequence of four steps:

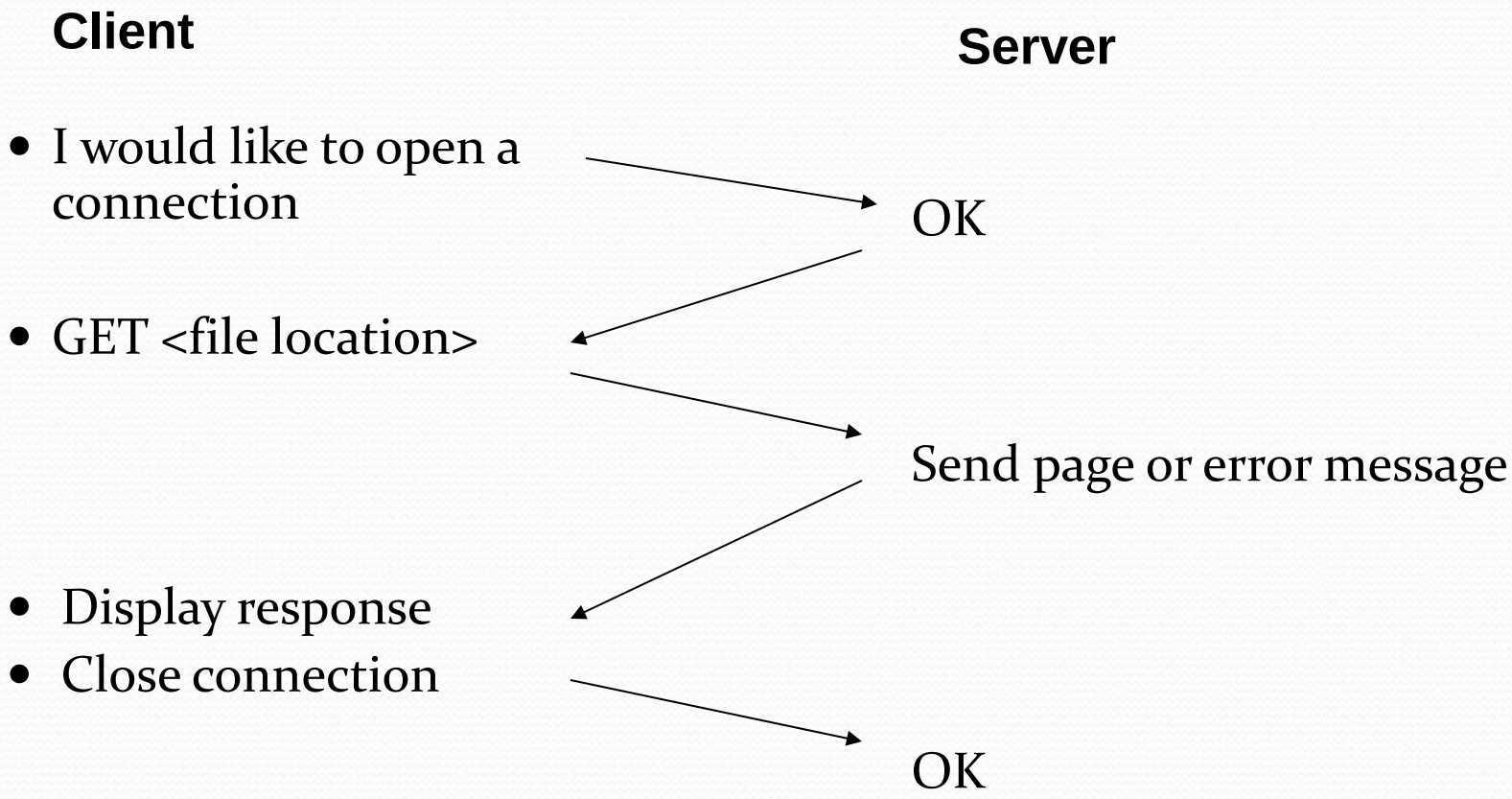
1. The client opens a TCP connection to the server on port 80, by default; other ports may be specified in the URL.

2. The client sends a message to the server requesting the resource at a specified path. The request includes a header, and optionally (depending on the nature of the request) a blank line followed by data for the request

3. The server sends a response to the client. The response begins with a response code, followed by a header full of metadata, a blank line, and the requested document or an error message.

4. The server closes the connection. This is the basic HTTP 1.0 procedure. In HTTP 1.1 and later multiple requests and responses can be sent in series over one TCP connection.

An HTTP conversation



HTTP is the set of rules governing the format and content of the conversation between a Web client and server

An HTTP example

The message requesting a Web page must begin with the work "GET" and be followed by a space and the location of a file on the server, like this:

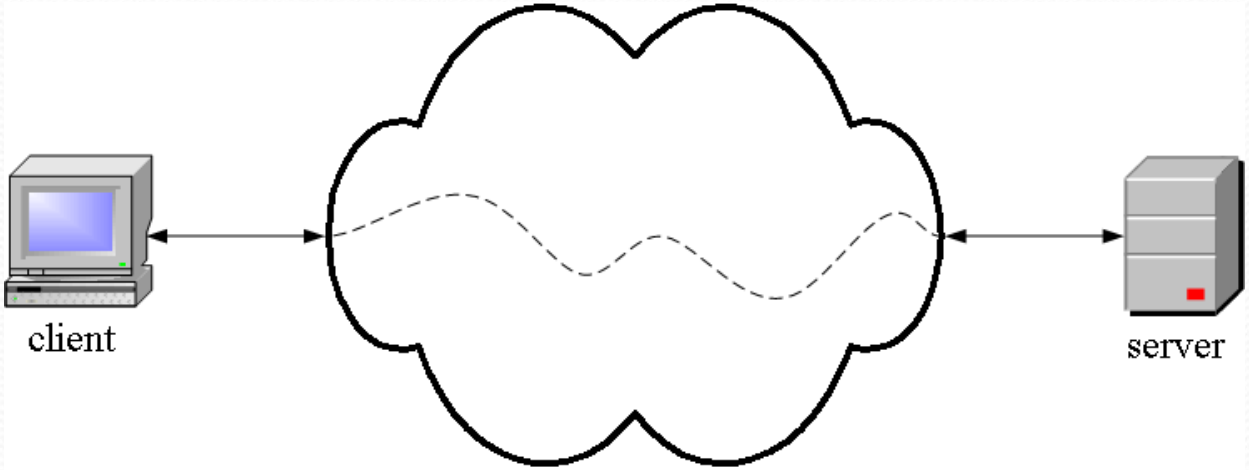
GET /fac/lpress/shortbio.htm

The protocol spells out the exact message format, so any Web client can retrieve pages from any Web server.

Network protocols

- The details are only important to developers.
- The rules are defined by the inventor of the protocol – may be a group or a single person.
- The rules must be precise and complete so programmers can write programs that work with other programs.
- The rules are often published as an RFC (request for comments) along with running client and server programs.
- The HTTP protocol used for Web applications was invented by Tim Berners Lee

HTTP is an application layer protocol



- The Web client and the Web server are application programs
- Application layer programs do useful work like retrieving Web pages, sending and receiving email or transferring files
- Lower layers take care of the communication details
- The client and server send messages and data without knowing anything about the communication network

The application layer is the boss-top layer

Layer	Function
Application	Do useful work like Web browsing, email, and file transfer
Lower layers	Handle communication between the client and server

- Your boss says: Send this package to Miami -- I don't care if you use Federal Express, UPS, or any other means. Also, let me know when it arrives or if it cannot be delivered for some reason.

- The application program says: Send this request to the server -- I don't care how you do it or whether it goes over phone lines, radio, or anything else about the details. Just send the message, and let me know when it arrives or if it cannot be delivered for some reason.

There are five TCP/IP layers, the application layer and four lower layers.

Many application layer protocols are used on the Internet, HTTP is only one

Protocol	Application
HTTP: Hypertext Transfer	Retrieve and view Web pages
FTP: File Transfer	Copy files from client to server or from server to client
SMTP: Simple Mail Transport	Send email
POP: Post Office	Read email

Brief History of HTTP

- 1990-1993: The idea of a web “browser” is contemplated – poor interfaces hinder browser use
- 1993: Marc Andreessen (then a grad student at NSCA) Nationa Strengths and conditioning association posts Mosaic on an ftp cite. New features include:
  - Hyperlinks
  - Embedded images
- December 1993: Mosaic growth makes the front page of New York Times
- 1994: Marc Andreessen and colleagues leave NSCA to form Mosaic Corp. (later renamed “Netscape”)

- A typical client request looks something like this:

- GET /index.html HTTP/1.1
- User-Agent: Mozilla/5.0 (Macintosh; Intel Mac OS X 10.8; rv:20.0)
- Gecko/20100101 Firefox/20.0
- Host: en.wikipedia.org
- Connection: keep-alive
- Accept-Language: en-US,en;q=0.5
- Accept-Encoding: gzip, deflate
- Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
- The first line is called the *request line*, and includes a method, a path to a resource, and the version of HTTP. The method specifies the operation being requested. The GET method asks the server to return a representation of a resource. /*index.html* is the path

- Each line takes the following form:
- *Keyword*: *Value*
- Keywords are not case sensitive. Values sometimes are and sometimes aren't. Both keywords and values should be ASCII only

HTTP Methods

- Communication with an HTTP server follows a request-response pattern: one stateless request followed by one stateless response. Each HTTP request has two or three parts:
- A start line containing the HTTP method and a path to the resource on which the method should be executed
- A header of name-value fields that provide meta-information such as authentication credentials and preferred formats to be used in the request
- A request body containing a representation of a resource (POST and PUT only)

- There are four main HTTP methods, four verbs if you will, that identify the operations
- that can be performed:
- GET
- POST
- PUT
- DELETE

The Request Body

- The GET method retrieves a representation of a resource identified by a URL. The exact location of the resource you want to GET from a server is specified by the various parts of the path and query string. How different paths and query strings map to different
- resources is determined by the server.
- POST and PUT are more complex. In these cases, the client supplies the representation of the resource, in addition to the path and the query string. The representation of the resource is sent in the body of the request, after the header. That is, it sends these four items in order:

1. A starter line including the method, path and query string, and HTTP version
2. An HTTP header
3. A blank line (two successive carriage return/linefeed pairs)
4. The body

- For example, this POST request sends form data to a server:
- POST /cgi-bin/register.pl HTTP 1.0
- Date: Sun, 27 Apr 2013 12:32:36
- Host: www.cafeaulait.org
- Content-type: application/x-www-form-urlencoded
- Content-length: 54
- username=Elliotte+Harold&email=elharo%40oibiblio.org
- In this example, the body contains an *application/x-www-form-urlencoded* data, but that's just one possibility. In general, the body can contain arbitrary bytes

- For example, here's a PUT request that uploads an Atom document:
- PUT /blog/software-development/the-power-of-pomodoros/ HTTP/1.1
- Host: elharo.com
- User-Agent: AtomMaker/1.0
- Authorization: Basic ZGFmZnk6c2VjZXJldA==
- Content-Type: application/atom+xml;type=entry
- Content-Length: 322
- <?xml version="1.0"?>
- <entry xmlns="http://www.w3.org/2005/Atom">
- <title>The Power of Pomodoros</title>

- <id>urn:uuid:101a41a6-722b-4d9b-8afb-ccfb01d77499</id>
- <updated>2013-02-22T19:40:52Z</updated>
- <author><name>Elliotte Harold</name></author>
- <content>I hadn't paid much attention to Pomodoro…</content>
- </entry>

Cookies

- HTTP cookies are data which a server-side script sends to a web client to keep for a period of time.

- On every subsequent HTTP request, the web client automatically sends the cookies back to server (unless the cookie support is turned off).

- The cookies are embedded in the HTTP header (and therefore not visible to the users).

Cookies

- **Shortcomings of using cookies to keep data**
  - User may turn off cookies support.
  - Data are kept with the browser
    - Users using the same browser share the cookies.
  - Limited number of cookies (20) per server/domain and limited size (4k bytes) per cookie
  - Client can temper with cookies
    - Modify cookie files, use JavaScript to create/modify cookies, etc.
- **Notes**
  - Don't always rely on cookies as the client may have turned off cookies support.
  - Don't store sensitive info in cookies

Cookies

- Usages:
  - Identifying a user during an e-commerce (or other) session
  - Avoiding user-name and password
  - Customizing a site
  - Focusing advertising

Cookies

- Cookies are state information that gets passed back and forth between the web server and browser in HTTP headers
  - A response header

```
Set-Cookie: NAME=VALUE; expires=DATE;
path=PATH; domain=DOMAIN_NAME; secure
```

  - A request header

```
Cookie: NAME=VALUE; NAME2=VALUE2; NAME3=VALUE3...
```

Problems

- A privacy threat:
  - search engine can remember previous searches
  - The computer that stores the cookie can allow an access to a site for a person that is not the person that the site recognizes

- However, cookies do not pose a security threat

Sharing Information

- Can two sites share the information that they have with cookies?
- What if the two sites use images from the same source?

Limitations on Cookies

- Must be set before any HTML is generated
  - Neither servlets embedded using the SERVLET tag, nor chained servlets, can set a cookie
  - They can still access cookie values
- Name isn't unique
  - Uniqueness enforced on (Name, Domain, Path)
    - Most applicable cookie of each name (best match) is returned
- Only name, value, and version are returned

Cookie Defaults

- Max Age – if not set, the cookie will expire when the browser is closed
- Domain/Path – together, they specify where the cookie is valid
  - A cookie created in response to the request

```
http://www.cs.huji.ac.il/~dbi/home/index.html
```

```
domain: www.cs.huji.ac.il/
path: ~dbi/home/
```

CookieManager

- **Java 5**
  - **Java.net.CookieHandler**
- **Java 6**
  - **Java.net.CookieManager**
  - **CookiePolicy.ACCEPT_ALL**
  - **CookiePolicy.ACCEPT_NONE**
  - **CookiePolicy.ACCEPT_ORIGINAL_SERVER**

Properties of Cookies

- **`getCookieStore`**
- **`getDomain`** / **`setDomain`**
  - The domain for which the cookie belongs
- **`getMaxAge`** / **`setMaxAge`**
  - How long (in seconds) will the cookie last
  - Negative value = per-session cookie
- **`getName`**
  - The name of the cookie to identify it

Properties of Cookies

- **`getPath`** / **`setPath`**
  - Defines the path for which the cookie relates
  - **`Cookie.setPath("/")`** means that all the pages on the server will get the cookie
- **`getSecure`** / **`setSecure`**
  - Should the cookie be sent with SSL secured line
- **`getValue`** / **`setValue`**
  - The value that the cookie holds

Some more Cookie methods

- public void setComment(String purpose)
- public String getComment()
- setSecure(boolean flag)
- public boolean getSecure()
  - Indicates to the browser whether the cookie should only be sent using a secure protocol, such as HTTPS or SSL