Lecture 5 – HTTP

HTTP or the hypertext transfer protocol is the protocol used for transferring data between web clients and web servers. It has been standardized by the organization responsible for Internet standards, *IETF* (Internet Engineering Task Force). Unlike XHTML and CSS which are still being extended and updated, HTTP has undergone little change in the last decade. The current version is HTTP 1.1 and it has been described in RFC 2068 from 1997 updated by RFC 2616 from 1999 (IETF standards are called *RFCs* and each has a number it is known by).

Structure of HTTP interactions

HTTP uses TCP (transmission control protocol) which is a reliable connection-oriented service supported by the Internet. After the TCP connection is set up, both ends can send data to each other. An application can send data on a TCP connection using the same methods as for writing to files, and it can receive data using the same methods as for reading from files. If either the client or the server crashes while the TCP connection is open, TCP signals an error at the other end.

TCP allows a computer to run multiple services. When handling an incoming connection request for such computers, TCP needs to determine which service the client wants to use. TCP uses *port numbers* to solve this problem. This solution is akin to the use of telephone extensions to route incoming calls for a large organization where the caller enters the extension to select the party he wants to talk with. By default HTTP uses port 80, but it is possible to instruct the web server application to accept connections on a different port. URLs can encode alternative port numbers after the server name. For example the URL `http://www.xyz.com:8080/file.html`

1. Browser initiates TCP connection to server on port 80
2. Server accepts connection
3. Browser sends through the connection an HTTP request with the URL of the document
4. Server receives request, reads web page from disk, and puts it inside a reply message it sends to the browser
5. Browser receives page, renders it, realizes that the page includes two images stored on the server
6. Browser sends server request for first image
7. Browser sends server request for second image
8. Server receives request for first image and sends reply with the image
9. Server receives request for second image and sends reply with the image
10. Browser receives first image and re-renders the page
11. Browser receives second image and re-renders the page
12. Browser closes connection
13. Server closes connection

Figure 1: Possible HTTP interaction

instructs the browser to open the TCP connection to server `www.xyz.com` on port 8080.

HTTP is a stateless request-response protocol. A separate request is made for each HTML document, style sheet, and image. While multiple requests to the same server can use the same TCP connection to avoid the overhead of setting up a new connection for each request, the requests themselves are handled independently by the server. Note that treating images and style sheets as separate objects as opposed to as components of the web page has advantages. If multiple pages on the same site use the same style sheet or common images (e.g. the logo of the company), the browser does not need to get them again from the server when a second page is visited, it can use a cached copy of the image or style sheet that it downloaded with the first page. The previous page has an example of a possible timeline for a browser using HTTP to download a web page that uses two images from a web server. Note that images on a web page can be on different servers than the HTML document itself, in which case the browser would use separate TCP connections.

HTTP requests

HTTP also prescribes the exact format for the request and reply messages used by the browser and the server. HTTP uses a simple and flexible text-based format. Requests consist of a request line, followed by a number of header lines and for some types of requests by the body of the request.

```
GET /~estan/examples/Bucky.html HTTP/1.1
Host: pages.cs.wisc.edu
User-Agent: Mozilla/5.0 (Windows; U; Windows NT 5.1; en-US; rv:1.8.1.6) …
Accept: text/xml,application/xml,application/xhtml+xml,text/html;q=0.9,…
Accept-Language: en-us,en;q=0.5
Accept-Encoding: gzip,deflate
Accept-Charset: ISO-8859-1,utf-8;q=0.7,*;q=0.7
Keep-Alive: 300
Connection: keep-alive
Referer: http://pages.cs.wisc.edu/~estan/examples/
```

Figure 2: An actual HTTP request for URL http://pages.cs.wisc.edu/~estan/examples/Bucky.html

The request line consists of the HTTP command, a URL, and protocol version separated by spaces. The two most important HTTP commands (also called HTTP methods) are GET and POST (to be discussed later). The GET command is for getting documents from the server and it is followed by header lines followed by two new lines (an extra empty line). Most header lines are optional and many are to be used only in special circumstances, but some are used quite often. For both HTTP requests and responses header lines often carry metadata (information about the document or the request) and

context information that helps clients and servers work together well despite a wide variety of configurations, versions, and supported capabilities. Figure 2: An actual HTTP request for URL http://pages.cs.wisc.edu/~estan/examples/Bucky.html shows an actual HTTP GET request with all the header fields (some truncated). Most of the header lines in this request convey information about the browser that allows the server to customize its reply. For example if a page is available in multiple languages, the server can pick the one that fits best the client's preference. The client also specifies the formats, character sets, and compression methods it prefers. The `User-Agent:` line tells the server about the specific implementation of the client. The server may have multiple versions of the page, each customized for a given browser and this field can be used to select between them. This field also presents a valuable source of information that allows the server to track what browsers are popular among the people interested in the site. The `Referer:` header line is an important source of information about how the site is used as it holds the URL of the document from which the user arrived to the document being requested.

HTML forms generate (field-name,field-value) pairs that can be submitted to the server using either GET or POST. These two commands submit data to the server differently and they are to be used in different situations. GET is for retrieving data, so it should be used when submitting data that is meant just to help the server locate the information one wants (e.g. the coordinates for the portion of a map one wants to view online). POST is for sending data that will result in a change at the server (e.g. submitting your credit card number to a shopping site or updating your email or snail mail address in your bank's database). HTTP requests using the GET command do not have a separate position for the (field-name,field-value) pairs. These pairs are appended to the URL portion of the HTTP command line as follows. The first part of the URL points to the program on the server that must handle the submitted data, and it is followed by a list of pairs of the form field-name=field-value. The pairs in the list are separated by the `&` character, and the URL is separated from the list by the `?` character. For example one can use the URL `http://maps.com/streetmaps?x=43N&y=89W&zoom=large` to submit the zoom level and the x and y coordinates to a fictitious map-vieweing site. Note that there is a problem when the field names or the field values contain special characters such as space (because URLs cannot contain spaces and the server interprets the word after the space in the URL as the version of HTTP to be used), `?`, `&`, `=`, newline, and a few other characters. The *URL encoding* rules mandate that the escape character `%` be used followed by two hexadecimal digits indicating the character being escaped. For example `%20` stands for space and `%3f` stands for the `?` character.

The POST command of HTTP is for sending data to the server. Unlike GET requests, POST requests also have a body that follows after the two new lines that end the header line portion of the request. HTML forms can choose whether the data the user fills in is submitted using GET or POST commands. The formatting of the (field-name,field-value) pairs is not significantly different for the two cases, but for the POST request they do not follow the URL, but appear in the body. The POST command is not restricted to using such pairs in the body of the request, any type of data can be submitted as long as the client and the program handling it on the server agree on how it should be handled. For

example many web services submit XML-encoded requests in the body of HTTP POST requests.

Another HTTP command is HEAD. It works exactly like GET, except that the server is not expected to return the actual document in response to it, but just the header lines that the reply would normally contain. It is often used to check whether a hyperlink is valid or whether the document it points to has been modified recently.

HTTP replies

The web server responds to HTTP requests with HTTP replies. These consist of the status line, header lines, and in most cases the body of the reply. The status line has the version of the protocol used by the server, a status code and a textual explanation of the meaning of the code. The possible codes and their meanings are specified in RFC 2616. The most common codes are `200 OK` which indicates the successful completion of the request and `404 Not Found` indicating that the server does not have a page with the URL in the request. Other codes indicate that the requested document was moved to another URL, that the user does not have permission to view the requested document, or various error conditions that may occur.

```
HTTP/1.1 200 OK
Date: Tue, 21 Aug 2007 22:34:19 GMT
Server: Apache
Last-Modified: Thu, 08 Feb 2007 01:24:28 GMT
ETag: "64018364-187-e77d8b00"
Accept-Ranges: bytes
Content-Length: 391
Keep-Alive: timeout=5, max=100
Connection: Keep-Alive
Content-Type: text/html

<TITLE>Bucky Badger's web page</TITLE>
<BODY>
…
</BODY>
```

Figure 3: The reply to the request from Figure 2: An actual HTTP request for URL

Most of the reply header lines are optional and it is not uncommon for two responses from different sites to have very different header lines. The `Last-Modified` header line specifies the date the document was last modified. This information is useful for deciding whether the cached version of a document is the most recent one or not. The `Content-Encoding` line specifies how the body of the message is encoded. It is not uncommon for web pages or XML documents to be compressed (using one of the compression methods the browser claims to support in the `Accept-Encoding` header line of the HTTP request). The `Content-Type` field specifies the type of the document in the body of the reply. The method for naming various types of documents is called MIME (Multipurpose Internet Mail Extensions) has been developed for email and later adopted by the web. MIME is described in RFCs 2045 and 2046. MIME types consist of a main type and a sub-type. The most common types are `text` (which includes `html` and `plain` as sub-types), `image` (which has sub-types such as `gif` and `jpeg`), `audio`, `video`, and `application` which

specifies in the sub-type the name of the application that should be used to interpret the document. Note that the browser does not rely on file name extensions to determine how to interpret the body of the document (but the web server often relies on them to determine what MIME type to set in the `Content-type` header line of the reply).

The firebug add-on allows you to see the actual HTTP requests and replies for your browser including the header lines. This is a useful debugging tool and it can help you understand more about how HTTP works in practice.

HTTP header lines used for authentication

HTTP also has header lines for user authentication. If the web server is configured to require authentication for a site, the reply for the first request from a user will not contain the document, but a code indicating that authorization is required. The browser pops up a dialog asking the user for a user name and a password. Next the browser re-submits the request with an `Authorization:` header line that contains the user name and the password (or for some authentication methods the digest of the password). If the password is correct and the user is allowed to see the page, the page is returned in the next reply from the server. HTTP is a stateless protocol (requests are handled independently), so the server does not keep track of the fact that the user already authenticated. Therefore the browser automatically includes the `Authorization:` header line with the user credentials in all subsequent requests to the site until it is closed. There is no explicit log-out action.

HTTP cookies can be used for many purposes, including confirming user identity. With any reply the server can include a header line of the form `Set-cookie: name=value`. In response the browser stores the cookie and includes in every future request *to the site that sent the cookie* a header line of the form `Cookie: name=value`. Cookies have an expiration time, they can be explicitly destroyed by the web site that created them, and there are options controlling whether they persist after the browser is closed. Even though there are limits to how large browsers allow the cookies to be, they can be useful not just in authentication, but also in storing user preferences, implementing a shopping cart and other artifacts that require state at the browser and in tracking user behavior. Depending on your point if view (user or site interested in understanding user behavior) this can be a good or a bad thing.

We will discuss the privacy implications of cookies in more detail in the last segment of the course. In the same segment we will also discuss HTTP headers used by various techniques for improving the performance of web transfers.