April 2 Lecture Transcript – HTTP, part 2

Slide 1:

In the previous lecture, it was explained why HTTP is a '**stateless protocol**'. The key advantage of HTTP being stateless is the fact that, as a result, an HTTP/Web server does not have to establish and maintain session information for each of its clients, which saves on the server's CPU and memory resources and improves its overall performance.

Unfortunately, there are also serious downsides of HTTP being stateless. For example, once a requested Web-page is downloaded, the TCP connection between the client and the server is closed (i.e., all information pertaining to this session are erased/lost), and any subsequent request coming from the same client will look 'completely new'. That is, the server will not be able to recognize that this client has been previously served …

Some practical problems that arise from this include:

a) With the use of 'basic' (stateless) HTTP protocol, the server cannot derive an accurate count of its first-time vs. returning visitors;
b) Furthermore, by not being able to recognize (i.e., identify) its returning visitors/users, the server will not be able to personalize/customize its responses to best meet the needs of these users;
c) Also, by not being able to recognize/identify its users, the server is not in a good position to (e.g.) deny service to users that have previously exhibited malicious behavior …

One way to enable 'client identification' on top of stateless HTTP protocol is by means of the so-called 'HTTP (Web) cookies'. A cookie is a small pieces of data that:

1) **a server sends to a client/browser;**
2) **gets stored inside the browser;**
3) **is later automatically included in all subsequent requests sent to the originating server.**

In that way, the originating server is able to uniquely identify the given client at every subsequent visit …

HTTP protocol enables the initial setup and subsequent exchange of 'cookies' by means of Set-Cookie and Cookie headers. (More about these headers on the subsequent slide.)

The final point to make here: In everyday language, cookies are sometimes referred as 'malicious' with a potential to harm the client machine. Such an 'interpretation' of cookies is incorrect! Cookies are (just) small pieces of data (not code or executable!) that simply get stored on the client machine, and then get exchanged with the respective servers. The 'maliciousness of cookies' does not refer to what they can do to the client machine/computer, but to the fact that they can be used for the purposes of user tracking, and as such are a big threat to user privacy.

Slide 2:

In this slide, the exchange of cookies by means of 'Set-Cookie' (which belongs to the group of Response Headers) and 'Cookie' (which belongs to the group of Request Headers) is illustrated.

Slide 3:

The figure provided in this slide illustrates the actual mechanism of how a cookie is set and then used to identify/recognize the user.

In the figure it is assumed that the user initially has only one cookie in its 'cookie file' – a cookie associated with eBay server/Web-site. The user (then) sends a first-time request to Amazon server. Given that this request does not contain any Cookie headers/value, Amazon server recognizes/marks this user as 'new'. Consequently, Amazon server performs the following steps:

a) it creates an 'entry' in its 'user database', in which it stores this user's information such as the user's IP address, type of browser and OS, etc.
b) it generates a new cookie 'value' for this user - **1678** (which is also stored in the user database)
c) it creates and sends a response, containing the (new) cookie value, back to the user …

Once the user receives the given response, it extracts the cookie value (**1678**) and stores it in its cookie file. All subsequent requests sent by this client to Amazon server will contain **1678** in the cookie header, which will allow the server to precisely (re)identify the user …

Slide 4:

The figures provided in this slide illustrate the fact that a server can set multiple cookies on the same client machine. And, the client and return multiple cookie values to the server using the same (one single) cookie header.

Slide 5 and 6:

These two slides show Wireshark captures of:

1) an HTTP response form amazon.com server in which two different cookies are returned to the client (see Slide 5)
2) a subsequent HTTP request sent to amazon.com carrying the two previously set cookies inside a single HTTP header (see Slide 6).

As an exercise, try to determine the nature of HTTP response on Slide 5. What does the Response Code '302' indicate happening?

Slide 7:

As indicated earlier, even though cookies are generally useful and can be used to 'customize' the service provided to individual users/clients, they can also be used for potentially malicious purposes. Some of the most common issues related to the use of cookies are:

1) Cookies are, ultimately, used for (re)identification/tracking of users; yet, users have no (easy) way of knowing or controlling when/whether cookies will be stored on their computer and used to track them. The exchange of cookie information is 'automatic' and happens 'in the background' hidden from the view of the user(s) …
2) Any, even the smallest object (e.g., 1x1 pixel image) existing in a Web-page could be used to trigger the 'injection' of cookies into the computers of the users visiting this page. In fact, the use of such third-party-owned 'tracking pixels', that are invisible to the human eye and often have nothing to do with the actual content of the visited Web-page, is very prevalent in todays WWW.

Slide 8:

Over the years, two main challenges of WWW have been:

a) how to reduce **page response times** (i.e., the amount of time it takes for a Web page request to arrive to the server plus the time it takes for the response to arrive back to the client and be displayed on the client's screen);
b) how to reduce the overall load on individual Web-servers.

One popular approach to dealing with both of those challenges is the use of Web Caches / Proxy Servers. As illustrated in the figure provided in the bottom of this slide, a Proxy Server is an intermediary entity/computer placed within a LAN, whose purpose is to receive and forward HTTP requests/responses on behalf of other LAN machines, and in the meantime also store (i.e., make local copies) of all obtained/returned Web-page responses. As a result, any subsequent Web-page request for which the Proxy contains a local copy, the Proxy is able provide an immediate response, without forwarding the request to the (remote) server. This clearly achieves both – a shorter page-response time for the client, and a reduced load for the end-server.

Note that most modern Web-browsers come with the option of an automated Proxy Server setup. (An example of this settings option in Chrome is shown in the upper-left corner of the slide.) So, in an organization/LAN that deploys a Proxy Server, the address and port number of that server should be entered in the provided field (of Chrome settings), on all of the LAN's machines. Subsequently, any request sent through any Chrome browsers on this network will (first) be forwarded to the company's Proxy Server …

Clearly, the use of Proxy Servers come with some disadvantages of their own. Those include:
1) Proxies can only provide what they have already cached; but, what they have in their cache may not be the most recent version of the (cached) objects.
2) If proxies do not contain a requested object in their cache, then the overall delay of obtaining that object is longer for the client, as the request has to go through the extra step of being received and processed by the proxy, instead of being sent directly to the server …

Slide 9:

In this slide, we look into a numerical example pertaining to the use (and potential advantages) of Web caching.

The example considers an organization whose LAN supports data rates of 10 Mbps. There are 15 Web-requests sent from this LAN to one particular (remote) server every second. All these requests are for objects of the same size – 100,000 bits. The data rate between the organization's gateway router and a router that connects the gateway router to the rest of the Internet (i.e., to the remote server's LAN) is 1.5 Mbps.

Our task is to calculate the **download utilization** of the link/network inside the given LAN, and the **download utilization** of the external access link (between the two routers) assuming there is no other traffic being exchanged.
Note, by 'download' utilization we mean utilization with respect to the traffic that is being sent from the server back to the clients. **Also recall: Link/network utilization is the ratio of the traffic being carried over a link/network in [bps] vs. the actual capacity of this link/network in [bps].**

By putting the provided numbers to work, we can easily see that:
- The utilization of the internal link/network is (only) 15%.

- The utilization of the outside/access link is 100%. Such a high utilization implies that, inevitably, there will be very large delays in the requested objects arriving back to the respective clients on the internal LAN. (In the slide, we assume that these delays can be in the order of minutes.)

Slide 10:

In order to minimize the delays observed in Slide 9, one option would be to (assume the) use of a Proxy Server. Here, we will assume that the 'hit-rate' of the Proxy Server (i.e., number of repeat-requests that can be successfully resolved by the Proxy Server) is 40%.

With the use of such a Proxy, the utilization of the outside/access link would drop from 100% down to 60%, which is a significant difference, and is likely to result in a significant reduction of the overall downlink delay. (We assume the new reduced delay on the access link to be 10 msec.)

In order to calculate the average delay experienced by the users in the internal LAN in this particular case, you need to keep in mind that 40% of users will be served by the Proxy, but 60% will still have to be served by the remote server. So, to calculate the average, you (in fact) have to calculate a 'weighted average'!

Slide 11:

It was previously indicated that one downside of using Web Proxies is the possibility that they store/serve old(er) copies of Web objects which (in the meantime) have been changed/updated.

In this last slide, we are touching upon the issue of how to ensure that the items in a Web Cache/Proxy are fresh (i.e., latest copies), while minimizing the overall load on both the Proxy and the remote servers. Clearly, a very primitive approach of making sure that the items stored in a Proxy are 'fresh' is by forcing the Proxy to (re)fetch these items from their respective servers frequently – e.g., every few seconds. Undoubtedly, this approach would result in a very large number of (potentially) unnecessary HTTP request and responses exchanged between the Proxy and the servers.

Fortunately, HTTP protocol comes with provisions that allow the Proxy to inquire about the status of a Web object. And, only if the object has been changed since the last time it has been downloaded by the Proxy, the server will actually sent/forward the object to the Proxy … These provisions are implemented in the form of Conditional-GET message (i.e., HTTP request with If-Modified-Since Header).

Note: the use of If-Modified-Since is not exclusive to Web Proxies – it can also be used by regular clients (browser) to minimize the number/volume of exchanged Web-objects.

Slide 12:

This slide illustrates the use of Conditional-GET with If-Modified-Since header.