March 31 Lecture Transcript – HTTP, part 1

Slide 2:

The topic of today's lecture is HTTP protocol. The world's most popular application/framework that deploys this protocol is WWW. So, we begin the lecture by giving a brief overview of WWW …

In formal terms, World Wide Web (WWW) is a "**distributed client-server repository of hypertext and hypermedia objects**". Now, let us clarify this definition and its key terms:

1) The term '**hypertext and hypermedia repository**' refers to the fact that the objects stored in and retrieved through WWW are 'connected' by means of virtual links/pointers, also known as '**hyperlinks**'. (Think of a Web-page and its embedded links which point to other Web-pages.) Consequently, for an object (e.g., a Web-page, image, audio or video file, …) to be a part of WWW the object needs to have a unique address which serves as its 'hyperlink anchor'.
   Object addresses in WWW are called **Universal Resource Locators** (**URLs**). The URL of a WWW object consists of three main components:
   - the name of the protocol that can be used to retrieve this object (http),
   - the full (DNS) name of the machine that hosts/stores the given object file,
   - the complete file path for the given object on the host machine.

Slide 3:

In this slide, examples of two different URLs are shown, and their three main components are highlighted.

Slide 4:

2) The term '**distributed client-server**' in the definition from Slide 2 refers to the fact that the machines hosting WWW objects run the so-called server programs/applications, and these server programs provide/send their objects to the end users on-demand by means of the so-called client programs/applications (also known as **browser**)
   A Web-client (browser) can access WWW objects from many different Web-servers, as illustrated in the provided figure.

Slide 5:

The figure provided in this slide illustrates the fact that a typical Web-page contains a number of embedded objects (images, javascripts, css, etc.), and each of these objects could potentially be hosted by/at a different Web-server. So, in order to render just this one page, the browser needs to contact and fetch objects from all those servers …

According to https://httparchive.org/reports/state-of-the-web, the average number of resources requested by (i.e., contained in a Web-page) is greater than 70 …

Slide 6:

As stated in Slide 4, '**Web browser**' is a term used to refer to the client programs/applications of WWW. These programs are rather complex in terms of the number of tasks that they have to perform as well as

the number of protocols and languages that they need to implement/deploy in order to successfully retrieve and render Web pages. Two most important of those languages / protocols are:

1) HTML – programming language used to write / construct Web pages.
2) HTTP – networking protocol that facilitates the retrieval (i.e., exchange) of Web objects between Web servers and Web clients.

Slide 7:

This slide provides a closer look into the structure and operation of a Web browser. As illustrated in the provided figure, the most important component of a Web browser is '**controller**'. The controller receives input (i.e., requests for Web documents/objects) from the users by means of a keyboard. The controller deploys **client protocols** to fetch the requested documents/objects from their host servers. And, finally, the controller engages different **language interpreters** in order to render the requested/fetched documents/objects and present them to the user.

Slide 8:

In the rest of this lecture, we will focus on the HTTP protocol which (as previously stated) is the key protocol for retrieval of Web objects …

First of all, two most fundamental features of HTTP protocol are:

- **HTTP is a stateless protocol**. This means that each HTTP request is a 'stand alone' request, and it is not related (i.e., carries no reference) to the requests that come before or after it. From the server's perspective, this also means that Web servers do not maintain any 'state' or 'connection' information that would allow them to correlate requests coming from the same client. (Recall, in contrast, TCP was an example of stateful protocol …)
- **HTTP is an application-layer protocol**, and it relies on TCP for transport-layer support. Again, recall, if a Web-client wants to sent an HTTP request to a Web-server, the client needs to first establish a TCP connection with the server on the server's port # 80. Only after the connection has been established, the client can actually send the HTTP request …

Slide 9:

In this slide, a summary of the main features and differences between **stateful** vs. **stateless** protocols is provided.

Slide 10:

In this slide, a more detailed look at the actual steps of Web-client/Web-server communication is presented. It is clearly shown that prior to the first HTTP request (GET) being sent to the Web server, the client and server had to establish a TCP connection (i.e., had to go through 3-way handshake).

Slide 11:

Our discussion about HTTP protocol (in the remainder of this lecture) will focus on the version of this protocol that is currently most widely used in the world – HTTP/1.1.

It is important to note, however, that HTTP protocol has undergone 'evolution' since its original inception in 1991 – as shown in the upper-left figure. Also, as this figure shows, currently there exist two 'newer' versions of HTTP – HTTP/2 and HTTP/3.

While HTTP/1.1 is supported by 100% of today's Web servers and clients/browsers, according to the most recent data from https://w3techs.com/, HTTP/2 is supported by about 43% of Web servers/sites (see the lower-right figure). And, according to the same site, HTTP/3 is supported by only 5% of today's Web servers/sites.

Slide 12:

In this slide, a clarification about the motivation for HTTP/2 is provided. Also, it is clarified that HTTP/2 is not an (entirely) new protocol, but rather an improvement of the existing HTTP/1.1.

For those interested in learning further about HTTP/2, please follow the link provided in the slide …

Slide 13:

Before we proceed with the discussion about HTTP/1.1. in more depth, let us do one practical exercise here. In particular, let us consider the question posed in this slide: *How could you discover which particular version of HTTP protocol has been used by your browser to retrieve a Web-page and its embedded content/objects?*

One thought/approach could be to use Wireshark, and sniff all the traffic exchanged between your browser and the server hosting the given page. Clearly, some of the captured packet would be the actual HTTP packets exchanged between the two, and the headers of these packets would carry the information about the 'protocol version' …

Note, however, that the Wireshark approach will work only if the Web server hosting the given page does not use HTTPS - i.e., if the payload of the TCP packets exchanged between the sever and your browser are not encrypted. YorkU's server is one such server (unfortunately!), so you could use the Wireshark approach to answer the given question for the page/URL shown in this slide (www.yorku.ca).

However, many other sites do use HTTPS, and for them the Wireshark approach would not work!!!

Slide 14:

Another more universal approach to answering the question from Slide 12 – an approach that would work for any Web-site, even if the site used encryption – would be to deploy the 'Developer Tools' option in your Chrome browser. This option can be accessed by going to your Chrome's Settings -> More Tools -> Developer Tools, and then in the Developer Tools selecting the Network tab. (See the upper red arrow in the provided figure.) The Network tab shows the list of all objects that have been retrieved in order to render the given Web-page, and for each object various information is provided (e.g., type of object, initiator = which other object has initiated the retrieval of this object, object size, etc.) including the version of HTTP protocol that was used to retrieve this object. You can see that in the case of www.yorku.ca, most objects were retrieved using HTTP/1.1 (see the lower red arrow).

Slide 15:

With this slide we begin the in-depth discussion about HTTP/1.1 protocol. The first topic to discuss are two different modes of HTTP protocol operation – non-persistent vs. persistent.

Slide 16:

In the case of a **non-persistent** 'HTTP connection', the client established (and closes) a separate TCP connection for each retrieved object in a requested Web-page, as illustrated in the provided figure.

Note, here we use the term 'HTTP connection' to refer to the entire sequence of HTTP requests and responses that need to be exchanged between the client and all involved server(s) in order to obtain/download the requested Web-page with all of its embedded content. Typically, the first request/response will acquire the main HTML document. In this document the browser will identify a number of embedded objects, which will then be requested one by one …

Clearly, in the case of non-persistent HTTP, if a Web-page contains N embedded objects (in addition to the main HTML file), then the browser needs to establish/close (N+1) TCP connections – even if the main HTML file and all of the embedded objects are hosted by the same server!

In terms of the actual time required to retrieve such a page (with the main HTML file + N embedded objects) using non-persistent HTTP – we should keep in mind that for **each object, the actual establishment of a TCP connection takes a full round-trip-time (RTT) amount of time, and the time to request an object on 'top of' this TCP connection and retrieve the response is another (RTT + object-transmission) amount of time**. (See the provided figure.) Now, if for simplicity we consider all objects to be of the same size and resulting in the same object-transmission time O, then the overall time to retrieve the given page is:

$$(2RTT + MF_{transmission\text{-}time}) + N \times (2RTT + O)$$

(Note: the first part of the above expression is the time required to retrieve the main HTML file! Once the main file is retrieved and all N objects identified, the objects are then fetched one by one …)

Clearly, non-persistent HTTP approach – especially in cases where the main HTML file and its embedded objects are all stored/hosted on the same server – is very ineffective.

Though, keep in mind that for cases where the embedded objects are stored on <u>different</u> servers, non-persistent HTTP connection is the only possible way to go …

Slide 17:

In the case of **persistent HTTP**, and assuming a Web-page with all/many embedded objects being stored/hosted on the same server, the same TCP connection is used to retrieve all these objects. That is, the client and server have to go through only one 3-way handshake (connection establishment), and this one connection is used/maintained to retrieve all the embedded objects … (See the provided figures.)

Note, however, that persistent HTTP comes in two different 'versions':

- **Without pipelining** – in this case one embedded object has to be requested and (fully) retrieved before the request for the next object can be sent to the server.
- **With pipelining** – in this case the requests for multiple embedded objects can be sent at the same time.

Clearly, out of the two, the 'pipelining' version results in an overall shorter page-retrieval time.

Slide 18:

In this slide we are asked to derive the expressions for the overall page-retrieval time assuming non-persistent HTTP vs. persistent HTTP with pipelining. The size of the main HTML file and all embedded objects is assumed to be the same (O [bits]), thus resulting in the same file/object transmission time of O/R [bps].

The first provided expression – expression (a) – is equivalent to what we have already seen/derived on slide 16.

To understand the second provided expression – expression (b) – consider that the establishment of the TCP connection takes RTT times, the retrieval of the main HTML file (which must be done first in order to identify all the embedded objects) takes another RTT + O/R time, and then the actual retrieval of the embedded objects takes RTT + M*O/R time.

Slide 19:

The next topic to discuss is the actual formatting of HTTP messages.

HTTP protocol makes provisions for 2 different types of messages: requests and responses. Both types of messages are written in the plain-text human-readable form. The structure of the two types of messages is shown in the provided figure.

Slide 20:

The main **Request Line** in HTTP request messages has 3 significant fields:

1) **Request Method**: could be GET, HEAD, POST, etc., and it specifies the actual nature of this request (e.g., whether the client wants to retrieve an entire document - in which case the method would be GET, or the client wants to obtain just some meta-data about this document – in which case the method would be HEAD).
2) **Path**: which is the requested object's URL with the server's symbolic name removed.
3) **The version of HTTP protocol used**.

Slide 21:

The figure provided in this slide shows an HTTP request message captured and displayed in Wireshark. As an exercise, try to determine which resource/object/page has been requested by this request, and from which server?

Slide 22:

The main **Status Line** in HTTP response messages also has 3 significant fields:

1) **The version of HTTP protocol used**.
2) **Status Code**: a numerical 3-digit value used to describe the outcome of the respective request and/or nature of this response. For example:
   - codes that start with 2 (2xx) imply that the request was successfully received, understood and accepted, and this message carries the response
   - codes that start with 3 (3xx) imply that further action must be taken in order to complete the request
   - codes that start with 4 (4xx) imply that the request contained incorrect syntax or cannot be fulfilled for some other reason
   - codes that start with 5 (5xx) imply that the server has failed to fulfill a valid request.
3) **Status Phrase**: simple textual interpretation of the status code.

Slide 23:

The figure provided in this slide shows the Wireshark captured HTTP response corresponding to the HTTP request shown in Slide 21.

Slide 24:

HTTP request and response messages also contain different types of headers, which are generally used to exchange additional information between the client and the server. All headers are formatted following the same convention – the header name followed by a colon sign and a space and then the actual header value.

There are 4 main groups of headers:
General Headers – could appear both in HTTP requests and responses
Request Headers – appear only in HTTP requests
Response Headers – appear only in HTTP responses
Entity Headers – appear both in HTTP requests and responses

General Headers are used to provide some additional general information about the respective message. Note, these headers are primarily used to communicate information about the message itself, as opposed to what content they carry. They provide general information on how this message should processed and handled. Some examples of this type of headers are shown in the slide.

Slide 25:

Request Headers allow the client to provide information about itself to the server and/or give additional details about the nature of the request that the client is making. Some examples of this type of headers are shown in the slide.

Response Headers provide additional data that cannot be placed in the status line as well as to give information about the server. Some examples of this type of headers are shown in the slide.

Slide 26:

Entity Headers provide information about the actual content/resource carried in this message. They serve the overall purpose of conveying to the recipient of a message the information it needs to properly process and display the resource/entity, such as its type and encoding method.

Slide 26:

The figure shown in this slide illustrates the use of HTTP request and response messages and their respective headers.